
# PROYECTO FINAL DE CARRERA

**TÍTULO:** EspiroQ: APLICACIÓN PARA EL CONTROL DE CALIDAD DE ESPIROMETRIAS

**AUTOR:** DAVID FONOLLOSA BERLANGA

**PROYECTO FINAL DE CARRERA DE:**
INGENIERÍA TÉCNICA DE TELECOMUNICACIONES, ESP. SISTEMAS ELECTRÓNICOS.
**EPSEVG**

**DIRECTOR:** SERGIO SÁNCHEZ LÓPEZ

**DEPARTAMENTO:** 701, ARQUITECTURA DE COMPUTADORS

**FECHA:** FEBRERO DEL 2010

**Este Proyecto tiene en cuenta aspectos mediambientales: □ Si □ No**

**PROYECTO FINAL DE CARRERA**

**RESUMEN (máximo 50 lineas)**


La espirometría es una prueba de función pulmonar vital en la detección de enfermedades pulmonares obstructivas crónicas (EPOC).
Diversos estudios concluyen que la correcta realización de espirometrías fuera del entorno hospitalario se traduce en un aumento de detecciones precoces de EPOC, esto conlleva una disminución del flujo de pacientes a los centros especializados y finalmente un ahorro económico para los centros sanitarios.

En este proyecto se ha desarrollado EspiroQ, una aplicación Web que permite realizar la certificación de calidad, por parte de un profesional especializado, de las espirometrías realizadas fuera de los laboratorios de función pulmonar.

El proyecto se ha definido en dos fases: diseño y desarrollo.

Tras una pequeña introducción del término *espirometría* explicando sus características se ha iniciado la fase de diseño. El diseño se ha iniciado estudiando el prototipo desarrollado para el estudio Spir@p.
Posteriormente se ha realizado un pequeño estudio analizando diversos entornos de programación, que ha determinado que el lenguaje de programación que se adapta mejor a las necesidades del presente proyecto es Ruby y su entorno de desarrollo Web Rails (Ruby on Rails). Finalizando la fase de diseño se ha creado el modelo de datos de la aplicación.

La fase de desarrollo se ha llevado a cabo programando el modelo de datos, la lógica de negocio, los controladores de la aplicación y finalmente creando la parte gráfica con sus plantillas HTML[i] y hojas de estilo CSS[ii].

Se ha concluido el presente proyecto mostrando un ejemplo real y presentando las conclusiones finales


**Parlabras clave (máximo 10):**


| | | | |
|---|---|---|---|
| RUBY | RAILS | ESPIROMETRIA | CONTROL DE CALIDAD |
| HTML | MVC | WEB | |


[i] HyperText Markup Language (Lenguaje de Marcas de Hipertexto)

[ii] Cascading Style Sheets (hojas de estilo en cascada)

# AGRADECIMIENTOS

Al laboratorio de función pulmonar del Hospital Clínic de Barcelona, Linkcare y todos los colaboradores que me han enseñado todo lo necesario para poder realizar este proyecto.

De todos mis años como universitario a Salva, que aunque no le vea en años sé que siempre estará ahí para darme esas clases de Flash, CSS y diseño Web.

A "*les noies maques*" Núria, Marie y Andrea que gracias a su alegría, y sus cafés, consiguen que llegue y me vaya de la oficina con una gran sonrisa.

Pero en especial quiero dedicárselo a Marcela, que siempre ha estado y estará, a mi lado para apoyarme en los momentos buenos y malos, en la salud y en la enfermedad,…

Y sobre todo quiero agradecérselo a mi familia, que como me dijo un amigo, aunque la familia no se escoge, de todas las familias del mundo siempre me hubiera quedado con la mia. Con cariño para mi madre Rosi, y mis dos hermanas Maria e Isabel y "encara que no puguis estar aquí, moltes gràcies papa".

# ÍNDICE

1 INTRODUCCIÓN ..... 12

1.1 La espirometría ..... 12

1.2 Motivación ..... 12

1.3 Estado del arte ..... 15

1.3.1 NDD EasyWare Software ..... 15

1.3.2 W20 de Sibelmed ..... 15

1.3.3 Dataflow Management Software ..... 16

1.3.4 E-Spiro ..... 16

1.4 Objetivos ..... 16

1.4.1 Objetivos del proyecto ..... 16

1.4.2 Objetivos Personales ..... 17

2 DISEÑO ..... 18

2.1 Punto de partida ..... 18

2.1.1 E-SPIRO ..... 18

2.2 Fases de Diseño ..... 20

2.3 Requisitos específicos ..... 20

2.3.1 Usuarios de la aplicación ..... 20

2.3.2 Centros - Áreas Sanitarias ..... 21

2.3.3 Carga de espirometrías en formato XML en la aplicación ..... 21

2.3.4 Control visual de la espirometrías ..... 21

2.3.5 Control de evaluaciones ..... 22

2.3.6 Objetivos ..... 22

2.3.7 Estadísticas ..... 22

2.3.8 Exportación de datos explotados de la base de datos ..... 23

2.3.9 Integración con plataforma sanitaria Linkcare ..... 23

2.4 Decisiones tecnológicas ..... 23

2.4.1 Modelo de Espirómetro ..... 23
2.4.2 Tipo de aplicación ..... 23
2.4.3 Lenguaje de programación ..... 24
2.4.4 Entorno de desarrollo Web ..... 33
2.4.5 Base de Datos ..... 36
2.4.6 Servidor contenido dinámico ..... 37
2.4.7 Servidor contenido estático ..... 37
2.4.8 Sistema de control de versiones ..... 37
2.5 Patrones ..... 38
2.5.1 Principios de diseño ..... 38
2.5.2 Patrones de arquitectura ..... 38
2.5.3 Patrones de diseño ..... 40
2.6 Documentos de diseño ..... 41
2.6.1 Funcionalidades ..... 41
2.6.2 Flujo de navegación ..... 42
2.6.3 Modelos ..... 44
3 DESARROLLO ..... 50
3.1 Programación ..... 50
3.1.1 MVC ..... 50
3.1.2 Active Record (AR) Modelo ..... 51
3.1.3 Action Controller-Controladores ..... 54
3.1.4 ActionView (AV)-Vistas ..... 56
3.1.5 Sesiones y Autentificación ..... 62
3.1.6 Parseo XML ..... 63
3.1.7 Interpretación de las gráficas ..... 64
3.1.8 Representación tabular de los resultados de la espirometría ..... 68
3.1.9 Gráficas de Gestión ..... 72

3.2 Migración de datos ..... 74
4 CASO REAL ..... 75
5 CONCLUSIONES ..... 81
5.1 Conclusiones del proyecto ..... 81
5.2 Conclusiones Personales ..... 81
5.3 Trabajos futuros ..... 82
6 Bibliografía ..... 83

# Índice de tablas

Tabla 1- Roles y privilegios ..... 21

Tabla 2 - Comparativa Entornos desarrollo Web - Ruby ..... 35

Tabla 3 - Resumen líneas de código EspiroQ ..... 50

Tabla 4- Valores Tabla 1 - detalles espirometría ..... 69

# Índice de Ilustraciones

Ilustración 1 - Distribución de los pacientes estudiados según el nivel de atención ..... 14

Ilustración 2 - Datos de equipamiento de los centros de atención primaria y neumología. ..... 14

Ilustración 3 – jerarquías areas sanitarias ..... 21

Ilustración 4 - Índices aceptación lenguajes programación - Yahoo Search ..... 25

Ilustración 5 - Índices aceptación lenguajes programación - Google Code ..... 25

Ilustración 6 – Índices aceptación lenguajes programación – Freshmeat ..... 26

Ilustración 7 - Índices aceptación lenguajes programación – Ohloh ..... 26

Ilustración 8 - Índices aceptación lenguajes programación – Delicious ..... 27

Ilustración 9 - Índice aceptación lenguajes programación – Craigslist ..... 28

Ilustración 10 - Tendencia del crecimiento de la demanda L.P. - Indeed.com ..... 28

Ilustración 11 - Tendencia de la demanda L.P. - Indeed.com ..... 29

Ilustración 12 - Índice aceptación lenguajes programación - Global ..... 29

Ilustración 13 - Tendencia del crecimiento de la demanda Framework Ruby. - Indeed.com .... 36

Ilustración 14 - Arquitectura MVC ..... 39

Ilustración 15 – Arquitectura MVC en Rails ..... 40

Ilustración 16 - codigo EspiroQ - patron iterador ..... 41

Ilustración 17 - Modelo Base de Datos E-Spiro ..... 45

Ilustración 18 - Modelo Base de Datos EspiroQ ..... 47

Ilustración 19 - modelo MVC - aplicación ..... 50

Ilustración 20- código - migration spiromaneuvers ..... 51

Ilustración 21 – control de versiones- migrations ..... 52

Ilustración 22 - código - Spirometry relaciones modelo ..... 52

Ilustración 23 - código - Spirometry validaciones básicas ..... 53

Ilustración 24 - lista de métodos clase Spirometry ..... 53

Ilustración 25 - métodos clase Spirometry ..... 53

Ilustración 26 - código - Controlador – Profesionales ..... 55

Ilustración 27 - código - routing aplicación ..... 55
Ilustración 28 - Routing App - resumen. ..... 56
Ilustración 29 - imagen - layout 1 ..... 57
Ilustración 30- imagen - layout 2 ..... 57
Ilustración 31 - código - separación por capas ..... 58
Ilustración 32 - CSS - modelo de cajas ..... 59
Ilustración 33 - CSS - Clases ..... 59
Ilustración 34 - lista partials ..... 60
Ilustración 35 - código - Vista - uso de partials ..... 61
Ilustración 36 - código - Modelo User ..... 62
Ilustración 37 - código - filtro verificar sesión ..... 63
Ilustración 38 - código - validación Schema ..... 63
Ilustración 39 - código - extracto parseo XML - 1 ..... 64
Ilustración 40 - código - extracto parseo XML - 2 ..... 64
Ilustración 41 - código - decodificacion base 64 ..... 65
Ilustración 42 - código-transformación Byte a Signed Integer ..... 65
Ilustración 43- código - Deltadecompresión ..... 66
Ilustración 44 - código - preparación gráficas -1 ..... 67
Ilustración 45 - código - preparación gráficas -2 ..... 68
Ilustración 46 - detalle espirometría - tabla 1 ..... 69
Ilustración 47 - detalle espirometría - tabla 2 ..... 71
Ilustración 48 - tabla2 - codigo partials ..... 72
Ilustración 49 - tabla2 -código partial - FVC ..... 72
Ilustración 50 - metodos obtención espiros profesional ..... 73
Ilustración 51 - métodos obtención espirometrías centro ..... 73
Ilustración 52 - métodos obtención espirometrias subcentros ..... 74
Ilustración 53 - espirómetro NDD ..... 75

Ilustración 54 - EspiroQ - login técnico ..... 75

Ilustración 55 - EspiroQ - importación de datos de espirometría a través del archivo XML ..... 76

Ilustración 56 - EspiroQ - Valoración Visual de la espirometría ..... 76

Ilustración 57 - EspiroQ - login coordinador ..... 77

Ilustración 58 - EspiroQ - Evaluación de calidad ..... 78

Ilustración 59 – EspiroQ - Detalles generales de la espirometría ..... 79

Ilustración 60 - EspiroQ - Gráficas de gestión del Centro ..... 79

Ilustración 61 - EspiroQ - Gráficas de gestión del profesional ..... 80

Ilustración 62 - EspiroQ - Consulta de detalles de evaluación realizada ..... 80

# Anexos

A.A Código Fuente

A.A.a Controladores

A.A.b Modelos

A.A.c Vistas

A.B MIGRACIONES

A.C MAPEO DE PETICIONES HTML (ROUTES)

A.D schemasFinalXml.xsd

A.E EQUIVALENCIAS ENTRE TABLAS E-ESPIRO Y ESPIROQ

A.F MANUAL EXPORTACIÓN DATOS ESPIROMETRO NDD EASYONE

# 1 INTRODUCCIÓN

## 1.1 La espirometría

La espirometría es una prueba de función pulmonar que permite medir el volumen de aire que los pulmones pueden inhalar y exhalar en función del tiempo.

En la valoración de las enfermedades respiratorias, la espirometría tiene un papel de valor incalculable y se puede decir que aporta igual, o mayor, nivel de información que la que aporta una medición de tensión arterial sobre la salud cardiovascular general.

La espirometría permite al profesional detectar y medir la severidad de los síntomas de la enfermedad respiratoria, así como también permite realizar un seguimiento del progreso de la enfermedad para de esta forma establecer el mejor tratamiento de la enfermedad

La espirometría es una prueba básica para el estudio de la función pulmonar. Para garantizar su correcta realización e interpretación, las diferentes sociedades médicas neumológicas, nacionales e internacionales, han editado en los últimos años diversas recomendaciones y normativas que garantizan los mínimos necesarios para poder comparar sus resultados en cualquier parte del mundo.

La utilización de la espirometría para el diagnostico y evaluación del estado funcional se incrementa día a día. Dicho incremento se efectúa fundamentalmente en el ámbito de la Atención Primaria aunque se ha constatado una menor calidad con un impacto sustancial en los resultados y en la incorrecta interpretación de estos. Para ello deben fomentarse estrategias de asistencia remota que promuevan el uso extensivo de la espirometría forzada de calidad fuera de los laboratorios de función pulmonar.

## 1.2 Motivación

Hasta hace unos años las pruebas de función pulmonar, más específicamente las espirometrías, debido al poco conocimiento que se tenía de ellas no habían sido demasiado valoradas. Esta tendencia empezó a cambiar y actualmente las espirometrías tienen una importancia vital en la detección y tratamiento de numerosas enfermedades respiratorias. En la popularización de las espirometrías se pueden destacar tres hechos importantes:

La publicación internacional de un de consenso sobre la estandarización de la espirometría [1], y su posterior actualización del año 2005 [2]“*Standardisation of spirometry*” fue el primer factor importante.

El segundo hecho importante fue la estandarización de los valores de referencia de la población mediterránea, y por extensión de la población española “Spirometric reference values for a mediterranean population” [3].

Y el tercer factor y de mayor importancia para el propósito de éste proyecto, fue la ayuda de la industria para crear y administrar nuevos espirómetros al sector sanitario. La evolución de la tecnología en el diseño de aparatos de menor tamaño y mayor facilidad de uso fue determinante en la popularización de la espirometría.

El auge de la espirometría, sumado a los estudios que demuestran las incalculables ventajas, tanto económicas como sociales, que se obtienen del uso de los espirómetros en el diagnóstico, tratamiento y estudio de enfermedades, ha provocando que la realización de las espirometrías haya salido del entorno de los Laboratorios de Función Pulmonar y se haya extendido a otros entornos menos especializados en este tipo de pruebas. Esta salida del entorno hospitalario, como puede ser el de uso de la espirometría en los Centros de Atención Primaria (CAP) para el diagnóstico de trastornos respiratorios ha producido que la menor preparación de los técnicos se represente en una menor calidad de las espirometrías realizadas, por lo que se ha detectado un impacto considerable en los resultados y en la interpretación correcta de éstos. El estudio "*La espirometría en atención primaria en Navarra*" [4] confirmando lo dicho, concluye que la realización de espirometrías en la Atención Primaria la realizan diplomados en enfermería de los cuales tan solo el 64 % ha recibido la formación específica necesaria.

La realización de la espirometría aunque pueda parecer un proceso relativamente sencillo, en la práctica es bastante complicado hacerlo correctamente. Se puede decir que el proceso de realización de una espirometría no es tan sencillo como el de realizar una prueba de tensión arterial, lo que significa que la formación que ha de recibir el profesional sanitario que gestiona la correcta realización de las maniobras espirométricas es bastante compleja. Según la ATS[iii] [1]se considera que un profesional necesita 6 meses de formación para poder dirigir la práctica de espirometrías, con las suficientes garantías de que las maniobras se han realizado correctamente. Resaltando la dificultad que conlleva realizar una espirometría correcta y su posterior valoración para considerarla válida, también se puede citar la recomendación que hace la SOCIEDAD ESPAÑOLA DE NEUMOLOGÍA Y CIRUGÍA TORÁCICA, en su "Manual de procedimientos" [5] sobre la cualificación y preparación que ha de tener el profesional que dirige la espirometría:

- *Cualificación académica de diplomado de enfermería o similar.*
- *Habilidad en el trato con enfermos.*
- *Conocimientos de cálculo, electrónica, informática y representación gráfica de señales.*
- *Responsabilidad, hábito de toma de decisiones, capacidad de resolver problemas técnicos.*
- *Mínimo de 6 meses de entrenamiento en un centro reconocido.*

Para finalizar citar el estudio *"Fiabilidad del diagnóstico de la EPOC en la atención primaria y neumología en España. Factores predictivos"* [6] que dice que el uso de la espirometrías sirvió para diagnosticar 63% de los casos de EPOC de dicho estudio, donde el 11% fue en centros de Atención Primaria y el 51% en Neumología. De los centros de Atención Primaria únicamente el 49% disponía de espirómetro, el 30 % contaba con personal encargado de la prueba y tan solo el 22% de los centros realizaba controles de calidad periódicos.

[iii] American Thoracic Society

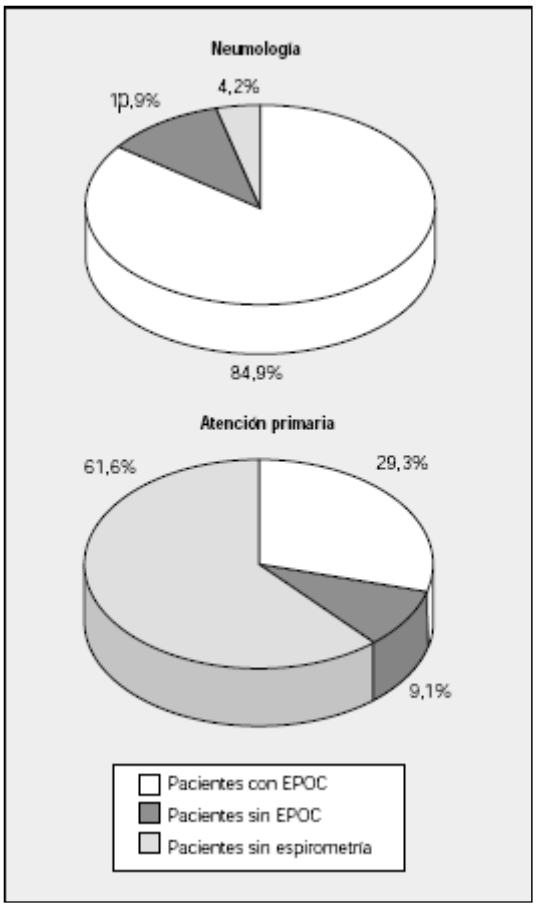



Ilustración 1 - Distribución de los pacientes estudiados según el nivel de atención

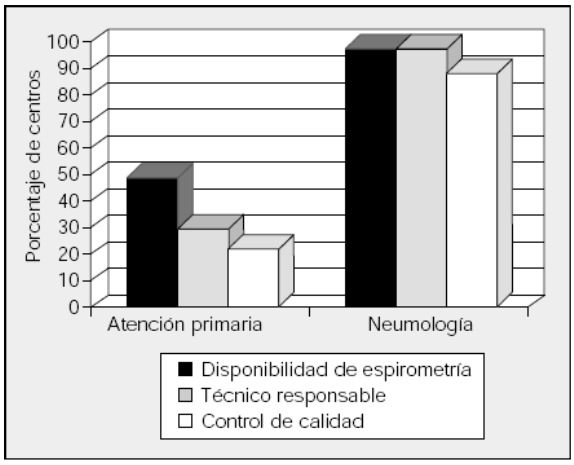



Ilustración 2 - Datos de equipamiento de los centros de atención primaria y neumología.

En conclusión realizar una herramienta que permita realizar un control de calidad remoto de las espirometrías repercutiría considerablemente en la calidad de éstas y en consecuencia se podrían diagnosticar muchísimo antes los síntomas de enfermedades pulmonares, permitiendo así realizar tratamientos preventivos. Todo esto se traduciría en mucha menos carga del sistema sanitario, y finalmente representaría un ahorro económico para la sanidad pública.

## 1.3 Estado del arte

Actualmente en el mercado hay diversos programas que permiten almacenar espirometrías para posteriormente visualizar los valores y gráficas resultantes, pero estos programas son específicos de las propias compañías de los espirómetros, las cuales aportan el software necesario para poder ver el resultado de las espirometrías realizadas con sus equipos.

El concepto de 'Control de Calidad Remoto de Espirometrías' es un concepto relativamente novedoso, por lo que en la actualidad, noviembre del 2009, en el mercado no se puede encontrar ninguna aplicación que permita visualizar, validar y controlar la calidad de las espirometrías que se han realizado.

La única herramienta conocida es E-Spiro, aplicación que fue diseñada para realizar la prueba piloto del proyecto E-Spir@p, el cual tenia entre otros objetivos los de evaluar la eficacia y los costes de un programa para evaluar la espirometría forzada en la Atención Primaria y analizar el impacto de una herramienta de tele-trabajo para potenciar la atención sanitaria en fases iniciales de la EPOC[iv]. Este proyecto fue dirigido desde el Hospital Clínic de Barcelona bajo la coordinación de DUE[v]. Felip Burgos.

Como software propio de los espirómetros se pueden encontrar por ejemplo:

### 1.3.1 NDD EasyWare Software

Software de la compañía *ndd Medizintechnik AG* [7] que permite la gestión del espirómetro NDD EasyOne en el ordenador. Esta herramienta es la que se utilizará para almacenar la espirometría realizada con el espirómetro NDD EasyOne para posteriormente generar los ficheros xml necesarios para importar a la aplicación web de control de calidad **EspiroQ**.

Algunas de las características de NDD EasyWare son:

- Vista preliminar de datos de medición y curvas.
- Introducción de datos del paciente en el PC
- Almacenamiento de mediciones en una base de datos local, compatible con Microsoft Access.
- Configuración sencilla del dispositivo EasyOne
- Presentación de curvas en tiempo real usando el conector de pantalla EasyOne.
- Exportación de los datos de medición de la base de datos a archivos de texto para el intercambio con otros programas.

### 1.3.2 W20 de Sibelmed

Software de espirometría SIBELMED [8], desarrollado bajo el entorno de Windows Microsoft, sus características principales son las siguientes:

[iv] Enfermedad Pulmonar Obstructiva Crónica

[v] Diplomado Universitario en Enfermería

- Permite la transparencia, análisis almacenamiento y/o registro de señales espirométricas.
- Gestión de diferentes tipos de bases de datos.
- Realización de las pruebas FVC[vi], VC[vii], MVV[viii]
- Presentación de gráficas en modo F/V (Flujo/Volumen) y V/T (Volumen/Tiempo)
- Gráficos de tendencias.
- Impresión de diferentes tipos de informes.

### 1.3.3 Dataflow Management Software

Software de gestión de los espirómetros Renaissance II [9], de la compañía Puritan Bennett. Entre sus principales características se encuentran:

- Mantenimiento electrónico de los registros médicos de los pacientes de función pulmonar.
- Verificación de la efectividad de tratamientos farmacéuticos mediante la evaluación de los datos de las pre y post sesiones.
- Archivado de datos en PDF para una posterior exportación.

### 1.3.4 E-Spiro

Herramienta experimental que permite el control de calidad y mejora de la calidad de las espirometrías realizadas en ámbitos diferentes del Laboratorio de Función Pulmonar como los Centros de Atención Primaria así como otros dispositivos sanitarios como la atención domiciliaria, Mutuas, Farmacias, etc., en particular permite certificar su calidad, corregir las desviaciones de los estándares y controlar el flujo de pacientes en los diferentes niveles asistenciales.

## 1.4 Objetivos

### 1.4.1 Objetivos del proyecto

El principal objetivo del presente trabajo de final de carrera es desarrollar una herramienta que permita realizar el control de calidad de espirometrías realizadas fuera de los centros de especializados, como puede ser Atención Primaria, farmacias, mutuas, etc.

Dicha aplicación ha de proporcionar los mecanismos necesarios para garantizar la correcta importación de las espirometrías y su posterior evaluación por parte de los profesionales de los centros de control.

[vi] Capacidad Vital Forzada

[vii] Capacidad Vital

[viii] máxima ventilación voluntaria

Se pretende además aportar las herramientas necesarias para que los gestores de los diferentes niveles sanitarios puedan gestionar la calidad de las espirometrías realizadas por los profesionales de las zonas de la que es responsable.

EspiroQ se ha de diseñar pensando en una futura integración con la plataforma de telemedicina Linkcare.

Ha de ser un producto el cual no sea demasiado costoso a nivel económico, por lo que un objetivo va a ser el de desarrollar en su mayoría mediante software libre.

El punto de partida de este proyecto va a ser una aplicación, E-Spiro, desarrollada en el año 2004 para una prueba piloto realizada en el Hospital Clínic de Barcelona. Se va a realizar un estudio de dicha aplicación y se va a intentar aprovechar la información recopilada en dicho estudio para realizar un análisis de la calidad de espirometrías por zonas sanitarias.

Con el objetivo de que la aplicación sea útil, y que cumpla con los requisitos actuales necesarios para poder realizar una buena evaluación y gestión de los resultados de las pruebas de esfuerzo pulmonar, se van a realizar pruebas junto con los usuarios reales, en este caso médicos, enfermeros y otros trabajadores sanitarios.

### 1.4.2 Objetivos Personales

A nivel personal surge la motivación de adquirir los conocimientos necesarios para poder programar una aplicación web 2.0 orientada al mundo de la telemedicina. En los momentos actuales de crisis el sector de la telemedicina tiene una clara proyección y muestra grandes posibilidades de inserción en el mundo laboral.

# 2 DISEÑO

## 2.1 Punto de partida

En el diseño de este proyecto se ha tomado como referencia la aplicación E-Spiro. Tal y como ya se ha comentado en el punto (1.3) E-Spiro nació en el marco de un estudio realizado entre otros por el grupo del departamento de neumología del Hospital Clínic de Barcelona liderado por el Dr. Josep Roca i por el DUE Felip Burgos. Dentro de este mismo grupo de trabajo es donde se está realizando el desarrollo de presente Trabajo de Final de Carrera por lo que la experiencia previa con la aplicación E-Spiro ha sido muy importante a la hora de desarrollar este proyecto.

El proceso de diseño poco estricto, generó una serie de problemas en la herramienta experimental, además de la aparición de nuevos requisitos surgidos a partir de la experiencia generada en el transcurso del piloto de puesta en marcha del estudio. Ha llevado a la decisión de rehacer completamente la aplicación. Se ha decidido cambiar las tecnologías, cambiar y añadir requisitos y modificar el diseño visual de la aplicación.

### 2.1.1 E-SPIRO

Aplicación desarrollada en 2004 de forma poco ortodoxa, se fue programando a partir de ideas, es decir que no se hizo ningún documento funcional, ni documento de requisitos, etc.

La idea original era realizar una aplicación en la que unos usuarios subían espirometrías y otros las puntuaban a nivel técnico y clínico.

#### *2.1.1.1 Arquitectura del programa*

El programa fue desarrollado usando una arquitectura típica de desarrollo de aplicaciones Web en Java, la cual está basada en los conceptos J2EE[ix], Arquitectura de tres capas y patrón de diseño Modelo-Vista-Controlador (MVC).

En esta arquitectura J2EE define un estándar para el desarrollo y ejecución de aplicaciones empresariales en lenguaje de programación Java. Utilizando Java EE para la programación se ahorró mucho tiempo de programación a bajo nivel ofreciendo más tiempo para pensar en los detalles de lógica de negocio y concepto.

La arquitectura de tres capas y el Patrón de diseño Model-View-Controller (MVC) son explicados más profundamente posteriormente en este mismo documento (2.5). Pero se señala que la tecnología utilizada para poder seguir dicha arquitectura fue:

Struts2 [10] como framework de desarrollo de aplicaciones web, Open Source y disponible para todas las plataformas J2EE. Dicho framework tiene como origen el patrón MVC (Modelo-Vista-Controlador).

Finalmente la tecnología utilizada para gestionar la persistencia de datos manejados por la aplicación fue Hibernate [11].

[ix] Java Platform, Enterprise Edition

#### 2.1.1.2 Puntos Fuertes

- **Idea**: La originalidad de la aplicación hace que este sea su principal punto fuerte. La no existencia de ninguna aplicación anterior similar a ésta y la tendencia de la medicina por disminuir gastos mediante el tratamiento fuera de los hospitales, hacen que el desarrollo de una aplicación que plasme fielmente la idea sea una oportunidad única.

- **Control visual de espirometrías online**: Dentro de la idea del control remoto de espirometrías, el poder consultar online las curvas y los datos resultantes es un factor único de la aplicación.

- **Gestión de diferentes dispositivos**: Finalmente se considera como otro gran punto fuerte que la aplicación E-Spiro era capaz de importar XML[x]s de tres dispositivos diferentes (NDD [7], Jaeger [12] y Sibel [8]).

#### 2.1.1.3 Debilidades.

Debido al no correcto diseño de la aplicación, se han detectado diversas debilidades.

- **Modelo de Base de datos difícilmente escalable:** La aplicación fue desarrollada utilizando un modelo de base de datos poco escalable. Con la intención de integrar la aplicación dentro de una plataforma de telemedicina llamada Chronic, se tomo el modelo de ésta como modelo de base de datos por lo que se acabó usando un modelo muy antiguo diseñado en los años 90, que no permitía una fácil evolución.

- **Desarrollo poco ortodoxo**: El desarrollo poco ortodoxo ha hecho que la aplicación se haya programado muchas veces a pedazos. Un ejemplo se puede encontrar en el modelo de datos, ya que si se observan las tablas se pueden ver algunas tablas con nombre en ingles y otras tablas con nombre en español demostrando que la aplicación se ha ido haciendo desestructuradamente.

- **Explotación de los datos:** La explotación de los datos es compleja ya que solo se puede realizar atacando directamente a la base de datos.

- **Imposibilidad de adquirir el conocimiento de la herramienta**: No se dispone de ningún tipo de código fuente ni documentación. Factor por el cual hace que evolucionar la herramienta sea algo muy complicado.

- **Tecnología obsoleta**: La imposibilidad de disponer del código fuente hace que no se puede migrar a nuevas versiones de la tecnología usada, por lo que es imposible evolucionar la aplicación y arreglar fallos de la aplicación.

- **Software propietario**: El hecho de trabajar con una base de datos Oracle, la cual requiere el pago de licencia para poder usarla hace que si algún día se quiere comercializar, el incremento de precio de la instalación pueda suponer un problema.

De las nuevas necesidades de los profesionales sanitarios se añaden nuevas debilidades a la aplicación:

[x] *EXtensible Markup Language (lenguaje de marcas extensible)*

- **No hay niveles sanitarios**: La aplicación gestiona solo un tipo de centro, por lo que no es posible realizar controles por diferentes niveles de distribuciones sanitarias del territorio.

- **Ausencia de funcionalidades de gestión**: No es posible ver estadísticas de niveles de espirometrías certificadas. Por lo que no permite, dentro de la aplicación, a los coordinadores realizar el control de calidad del trabajo realizado por los profesionales.

## 2.2 Fases de Diseño

El diseño de la aplicación de control de calidad de espirometrías se realizará a través de 3 fases diferentes:

1. *Requisitos específicos:* el primer paso en cualquier diseño es entender las necesidades concretas del cliente, este punto trata de ello.

2. *Decisiones tecnológicas:* la gran mayoría de decisiones tecnológicas se deberán tomar desde cero, ya que la antigua aplicación ha quedado muy desfasada en el tiempo.

3. *Documentos de diseño:* todas las ideas deben quedar plasmadas en estos documentos, que facilitarán el trabajo y la comprensión del código.

## 2.3 Requisitos específicos

A pesar de que partimos de una aplicación ya implementada, debido a la experiencia adquirida anteriormente por usuarios, los requisitos se han actualizado y ampliado.

### 2.3.1 Usuarios de la aplicación

La aplicación ha de ser capaz de gestionar diferentes tipos de usuarios. Según el rol con el que se acceda a la plataforma se podrá acceder a la totalidad de las funcionalidades o solo a las específicas del rol.

Actualmente hay definidos los siguientes roles

| Id | Nombre del Rol | Privilegios |
|---|---|---|
| 0 | Coordinador | - Valorar calidad espirometrías<br>- Visualizar espirometrías de todos los profesionales del centro.<br>- Establecer Alarmas del centro<br>- Todos los privilegios del profesional |
| 1 | Profesional | - Añadir nuevos pacientes<br>- Cargar nuevas espirometrías<br>- Visualizar espirometrías propias del profesional.<br>- Visualizar estadísticas propias del profesional + estadísticas |

| | | |
|---|---|---|
| | | globales del centro |

Tabla 1- Roles y privilegios

### 2.3.2 Centros - Áreas Sanitarias

La aplicación ha de ser capaz de gestionar diferentes tipos de áreas sanitarias. Donde cada área sanitaria tendrá a su vez una o más sub áreas. La jerarquía es la mostrada en la siguiente ilustración.

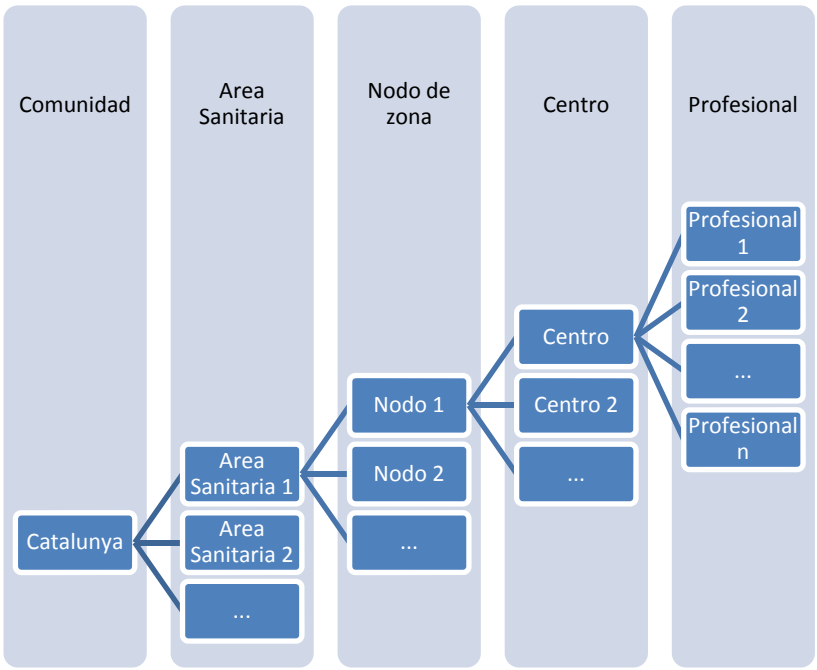



Ilustración 3 – jerarquías areas sanitarias

### 2.3.3 Carga de espirometrías en formato XML en la aplicación

La aplicación ha de poder leer los archivos XML[xi] generados por las aplicaciones de lectura de los diferentes espirómetros.

Han de existir mecanismos de validación de la correcta formación de los documentos XML y mecanismos de seguridad que aseguren que se están cargando los datos mínimos necesarios para poder realizar un correcto control de calidad.

En una primera fase se ha de poder leer los archivos generados por la aplicación Easyware de la compañía *NDD Medizintechnik*, en un futuro se considerará la integración de los espirómetros de las compañías *Jaeguer* y *Sibel*.

### 2.3.4 Control visual de la espirometrías

La aplicación ha de permitir realizar una verificación visual de la correcta realización de la maniobra espirométrica mostrando por pantalla los siguientes datos:

- Gráficas de las diferentes curvas de la maniobra.

[xi] *Extensible Markup Language (lenguaje de marcas extensible)*

- Valores calculados por el espirómetro:

  **FVC**: Capacidad vital forzada.(El mayor volumen de aire que puede ser expulsado de los pulmones en una maniobra forzada)

  **FEV1**: Volumen espirado en el primer segundo.(El mayor volumen de aire que puede ser expulsado de los pulmones en el primer segundo de una espiración forzada)

  **FEV1/FVC**: Relación entre el FEV1 y la FVC.

  **FEV6**: Volumen espirado en los 6 primeros segundos.(El mayor volumen de aire que puede ser expulsado de los pulmones en los 6 primeros segundos de una espiración forzada)

  **BE**: Volumen Extrapolado Anterior

  **EOTV**: Volumen al final del test

### 2.3.5 Control de evaluaciones

Tras la evaluación de la correspondiente espirometría a cargo del revisor, se ha de notificar automáticamente, mediante el envío vía correo electrónico de la evaluación, al profesional que realizó la espirometría.

### 2.3.6 Objetivos

Se ha de poder gestionar el número mínimo de espirometrías de calidad que cada zona sanitaria ha de realizar.

El coordinador será el encargado de fijar los objetivos y de realizar el seguimiento de ellos.

El no cumplimiento de los objetivos hará que se active una alarma visual que avisará al profesional y al coordinador que no se están cumpliendo.

El profesional podrá ver los objetivos del centro al que pertenece y cuál es el porcentaje de sus espirometrías de calidad.

### 2.3.7 Estadísticas

El modulo de estadísticas ha de proporcionar dos tipos de estadísticas diferentes, las relacionadas con la calidad de las espirometrías realizadas i las relacionadas con los datos sanitarios de dichas espirometrías.

Se han de poder consultar las estadísticas del profesional, centro, nodo, etc. Hasta llegar al nivel más alto de área sanitaria.

#### *2.3.7.1 Estadísticas sanitarias*

La realización de estadísticas sanitarias en las espirometrías que ya han sido “Certificadas” no es un requisito de una primera fase de desarrollo, pero se ha de contemplar que en un futuro la visualización de éste tipo de estadísticas serán de gran utilidad.

Este requisito queda pendiente de una futura definición de datos estadísticos necesarios para la realización de estudios.

#### *2.3.7.2 Estadísticas de calidad*

Las estadísticas sobre la calidad de las espirometrías permitirán realizar una importante labor de gestión de los profesionales. Permitiendo detectar problemas de formación. Las estadísticas de calidad se han de hacer de tanto de centros como de profesionales:

- Total de espirometrías.
- Total de espirometrías validadas.
- Número de espirometrías de calidad.
- Número de espirometrías no válidas.
- % sobre media del centro, área sanitaria.

#### *2.3.7.3 Unidades Territoriales Estadísticas –Nuts*

Como valor añadido, las exportaciones de los datos estadísticos referentes a zonas geográficas se han de exportar cumpliendo la **Nomenclatura de las Unidades Territoriales Estadísticas (NUTS)** [13], la cual es un estándar, creado por la Unión Europea, de geocodificación para referenciar las divisiones administrativas de los países para propósitos estadísticos.

### 2.3.8 Exportación de datos explotados de la base de datos

Los datos de la base de datos han de ser exportables, mediante la generación de un archivo de texto plano o una hoja de cálculo en formato Microsoft Excel.

### 2.3.9 Integración con plataforma sanitaria Linkcare.

Linkcare es una plataforma de aplicaciones de telemedicina desarrollada dentro de la *Fundació Clinic per la Recerca Biomédica* [14]. Las herramientas integradas en la plataforma han sido desarrolladas en un principio de forma independiente.

Así pues la plataforma Linkcare es modular (cada módulo es una de esas aplicaciones independientes), colaborativa (permite la colaboración de varios profesionales para un mismo paciente) y extensible (pueden añadirse tantos módulos como se necesite).

Una de las futuras herramientas de la plataforma ha de ser el modulo de control de calidad de espirometrías, por lo que la aplicación diseñada en este proyecto ha de estar preparada para ser integrada dentro de la plataforma Linkcare.

## 2.4 Decisiones tecnológicas

### 2.4.1 Modelo de Espirómetro

Se ha decidido que en la primera fase del proyecto se trabajaría con las espirometrías generadas por el espirómetro NDD EasyOne.

Esta decisión se ha consensuado con los profesionales sanitarios, ya que actualmente en el mercado es el espirómetro portable que toma las medidas espirométricas mas completas.

### 2.4.2 Tipo de aplicación

La experiencia adquirida con la aplicación E-Spiro ha demostrado que una aplicación Web es el mejor tipo de aplicación para el proyecto.

Como grandes ventajas se encuentran la facilidad de instalación, ya que solo es necesario disponer de conexión a Internet y tener instalado un navegador Web. La inmediatez de difusión del trabajo realizado, al ser una aplicación Web todo el trabajo que se realiza es automáticamente visible para todos los actores de la aplicación.

Las limitaciones propias de una aplicación Web, estándares entre navegadores (funcionalidad menor que la de las aplicaciones de escritorio, etc.) no han sido determinantes a la hora de utilizar la herramienta de espirometría por parte de los usuarios.

### 2.4.3 Lenguaje de programación

Un aspecto importante a la hora de realizar una aplicación Web es el de decidir que lenguaje de programación se ajusta mejor a las necesidades específicas del proyecto.

Actualmente se pueden encontrar multitud de lenguajes de programación, por lo que determinar cuál es el más adecuado para cada caso, puede resultar complicado. Por esta razón hay que tener una serie de factores en cuenta a la hora de realizar la selección.

En la elección se ha empezado por descartar todos aquellos lenguajes propietarios que su desarrollo requería el pago de cualquier licencia.

Tal como mencionan en “The Economics of Programming Languages” [15] a la hora de aprender un nuevo lenguaje de programación se han de tener en cuenta diversos factores como los siguientes: ha de ser fácil de escribir, ha de ser eficiente, ha de ser de alta calidad y ha de ser altamente productivo.

Actualmente existen centenares de lenguajes de programación y todos tienen sus pros y sus contras, por lo que llegar a analizar los factores anteriormente mencionados sería una tarea propia de un trabajo de final de carrera propio. Por lo que se ha decidido realizar un análisis de la relevancia de los principales lenguajes de programación dentro de la industria del software efectuando consultas a múltiples servicios con gran tráfico de usuarios, lo que servirá para determinar que lenguaje aporta mejores prestaciones.

#### *2.4.3.1 Estudio*

Para realizar el estudio se han evaluado, a través de la web *Programming Language Popularity* [16], servicios tan diversos como Craigslist [17],dice.com [18], indeed [19], Google Code [20], Ohloh [21], Delicious [22], Yahoo Search [23] y Freshmeat [24]. Posteriormente se ha hecho una valoración global de los resultados anteriores y finalmente se ha estudiado la tendencia a futuro de la programación.

##### 2.4.3.1.1 Yahoo Search:

Buscador de páginas web. Realizando la búsqueda genérica "*language* programming" se ha obtenido una clasificación con el número de referencias que hay en Internet de cada lenguaje.

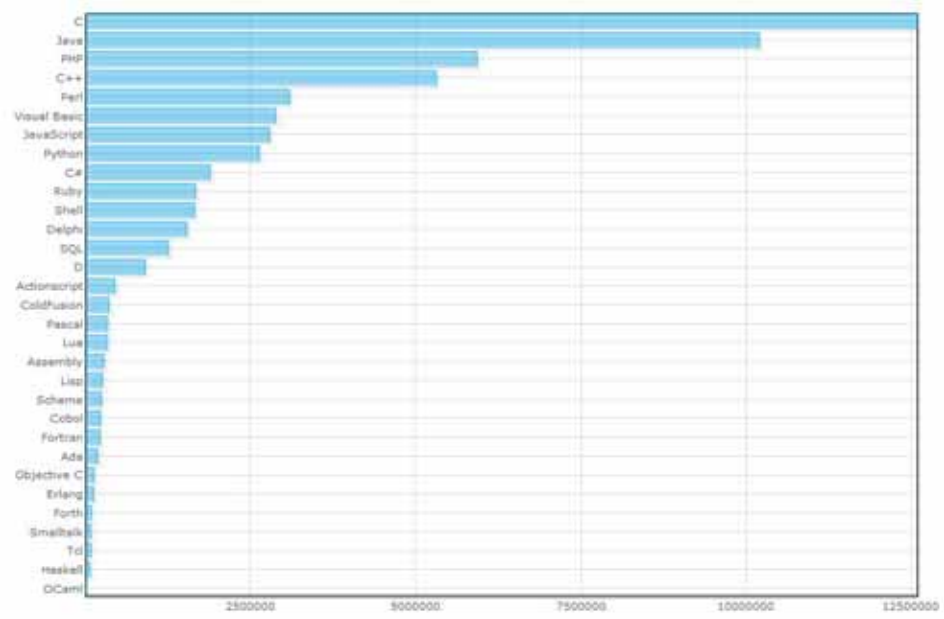



Ilustración 4 - Índices aceptación lenguajes programación - Yahoo Search

#### 2.4.3.1.2 Google Code, Freshmeat y Ohloh

Sitios web de referencia dentro de la comunidad de desarrolladores Open Source. Google Code y Freshmeat son servicios web donde se almacenan todo tipo de proyectos Open Source.

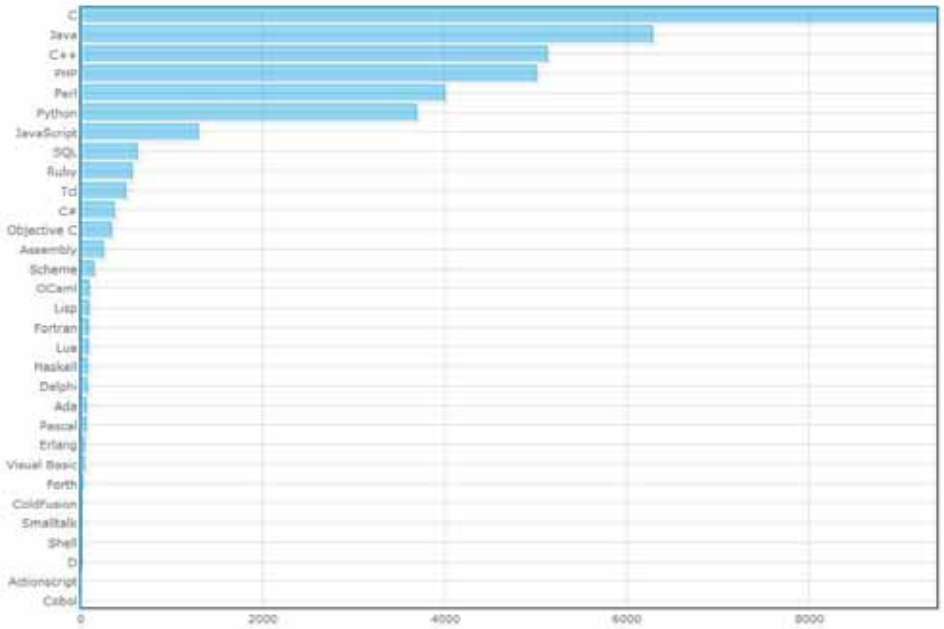



Ilustración 5 - Índices aceptación lenguajes programación - Google Code

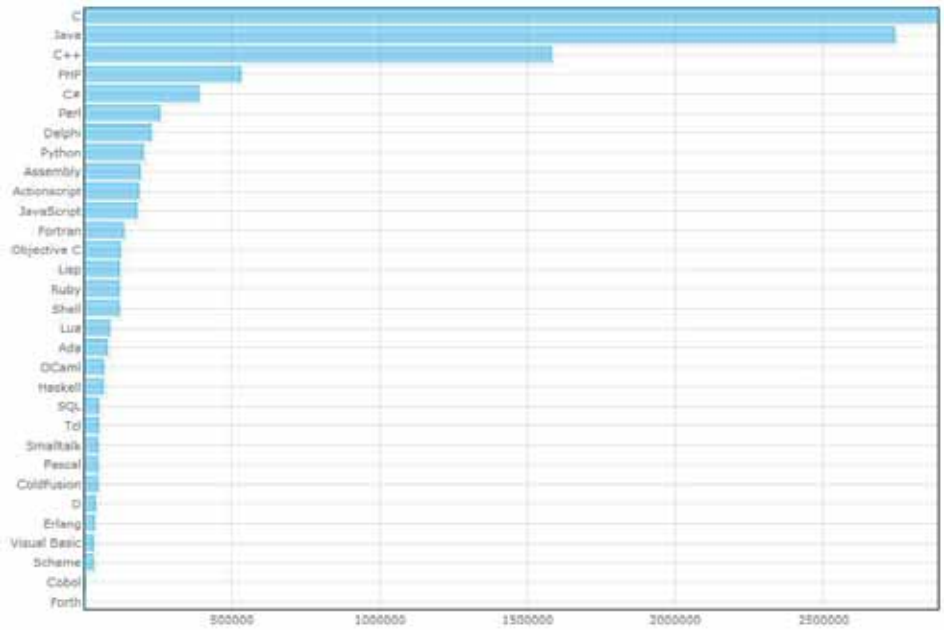



**Ilustración 6 – Índices aceptación lenguajes programación – Freshmeat**

Ohloh es un sitio web que tiene como objetivo intentar mostrar el estado del desarrollo de proyectos en código abierto, para su conseguir su cometido Ohloh provee de estadísticas e información diversa sobre varios proyectos opensource.

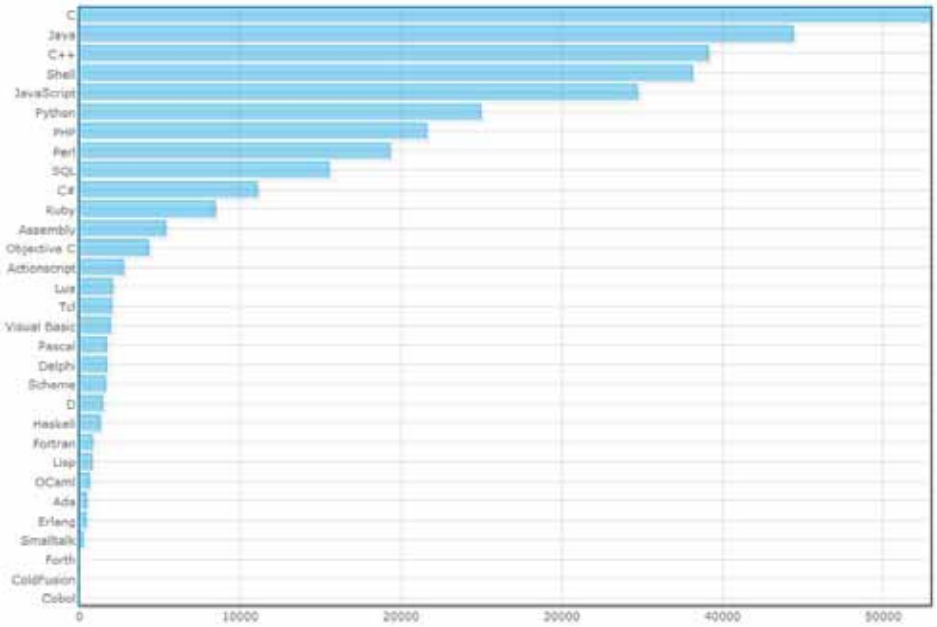



**Ilustración 7 - Índices aceptación lenguajes programación – Ohloh**

Las estadísticas que se han obtenido de estos servicios, no son significativas de las necesidades de la industria del software, pero si pueden indicar las preferencias de los programadores en el uso de un lenguaje u otro, ya que la elección de éste no vendrá determinada por quien pague, si no por las propias preferencias del programador.

#### 2.4.3.1.3 Delicious

Delicious es un sistema de marcadores sociales en web (Tags en ingles) con los que es posible categorizar la información contenida en las webs. Delicious no solo permite almacenar webs, sino que además permite compartir los marcadores con el resto de usuarios de Delicious.

Con el análisis del número de referencias a los marcadores referentes a los lenguajes de programación se han analizado los intereses de una comunidad de millones de usuarios.

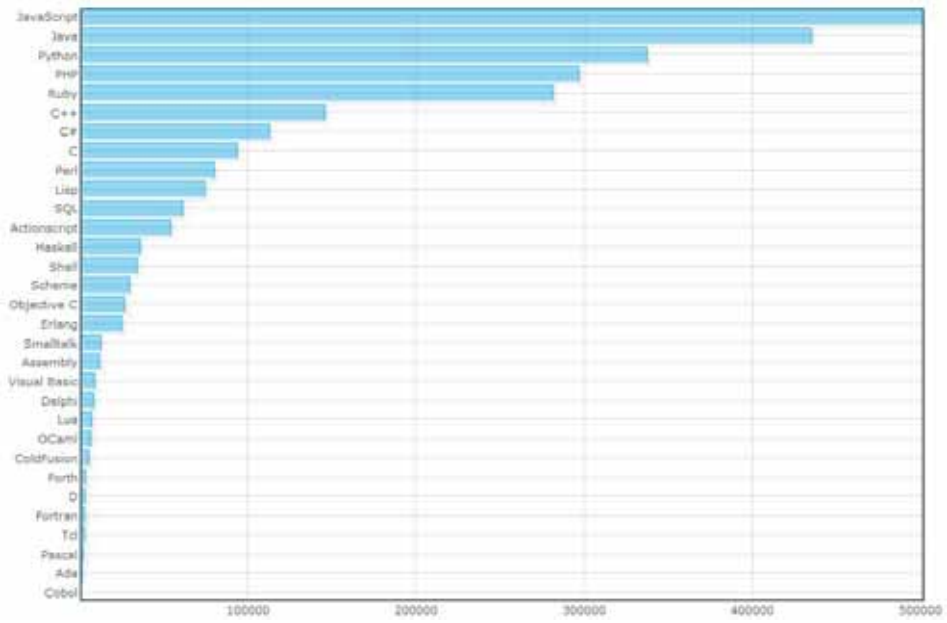



**Ilustración 8 - Índices aceptación lenguajes programación – Delicious**

.

#### 2.4.3.1.4 Craigslist, Indeed

Una de las mejores formas de evaluar las prestaciones y el grado de productividad de un lenguaje se obtiene a partir del nivel de aceptación de la industria del software. Esto queda claramente reflejado en el número de ofertas de empleo que demandan programadores con conocimientos de un lenguaje específico. Con este fin se han evaluado 3 populares sitios web.

Del sitio de Craigslist se han obtenido el número de ofertas de trabajo actuales.

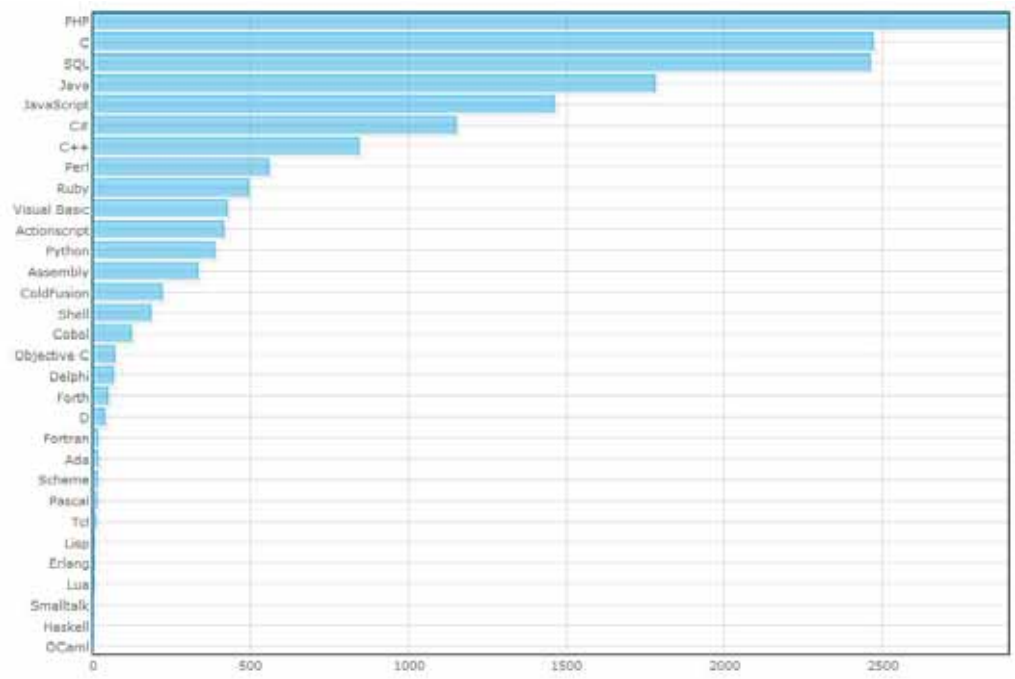



Ilustración 9 - Índice aceptación lenguajes programación – Craigslist

Finalmente del sitio web Indeed.com se ha realizado un pequeño análisis de las tendencias de la demanda de programadores con conocimientos específicos de un lenguaje de programación.

La primera estadística muestra la tendencia relativa de crecimiento. Donde se puede comprobar cómo en los últimos años la demanda, en general, se ha mantenido estable, a excepción de la necesidad de programadores Ruby que ha crecido considerablemente.

El gran crecimiento de Ruby, se puede considerar normal ya que es un lenguaje que es muy joven y está teniendo una gran aceptación

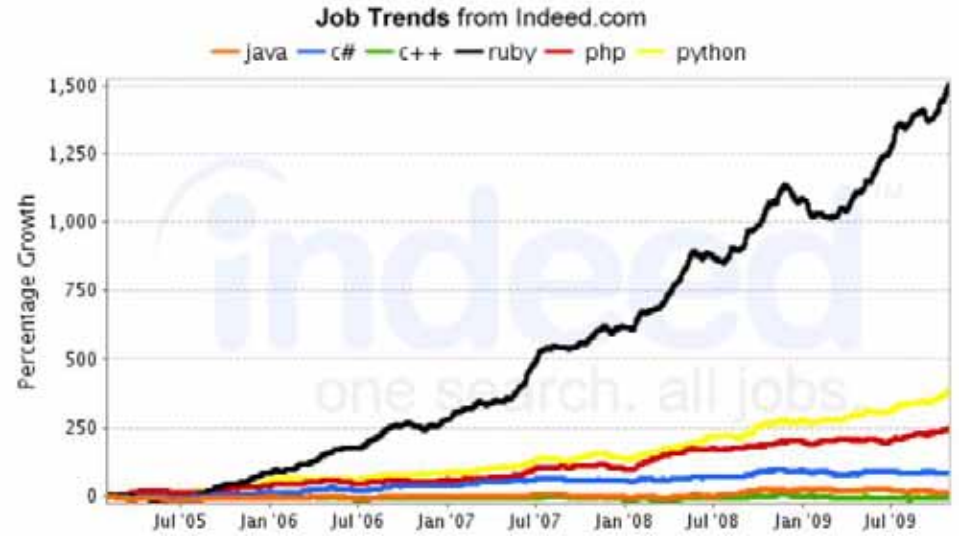



Ilustración 10 - Tendencia del crecimiento de la demanda L.P. - Indeed.com

En la segunda gráfica se puede ver la tendencia de la demanda en valores absolutos de los diferentes lenguajes. Aquí se puede comprobar claramente que Java es el lenguaje más demandado por el mercado en los últimos años. Esto no hace más que confirmar lo datos anteriores del estudio, los cuales indican que la comunidad de programadores se decanta claramente por Java como lenguaje de desarrollo.

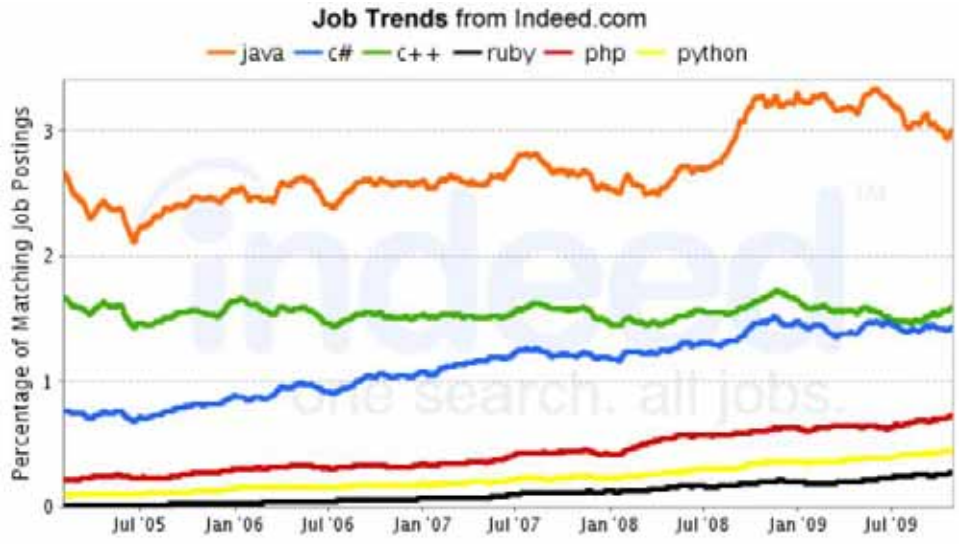



Ilustración 11 - Tendencia de la demanda L.P. - Indeed.com

#### 2.4.3.1.5 Conclusión

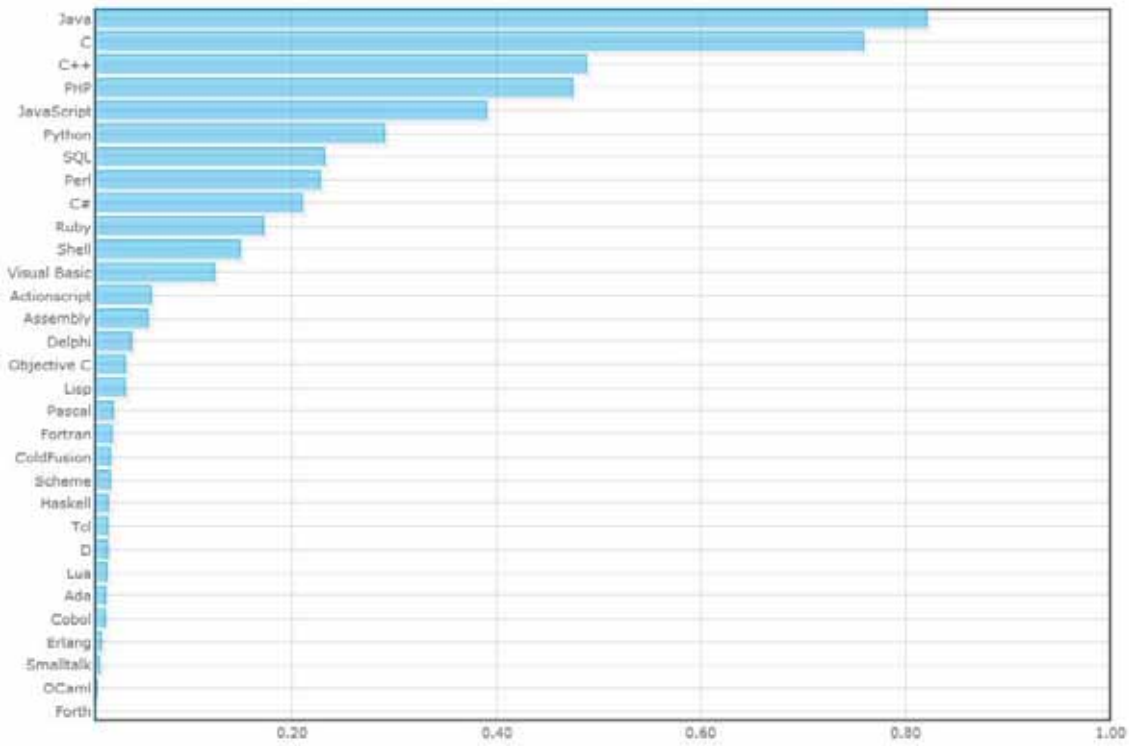



Ilustración 12 - Índice aceptación lenguajes programación - Global

Si se agrupan las estadísticas aportadas por las diferentes Webs analizadas y se toma como referencia la tendencia de ofertas de trabajo absolutas (Ilustración 9) se llega a la conclusión de que los lenguajes de programación con más popularidad son Java y C/C++.

C/C++ se pueden considerar lenguajes de programación de más bajo nivel que Java, lo cual aporta gran versatilidad y rendimiento a sus aplicaciones. La dependencia que a veces tienen sus librerías del sistema operativo, su complejo sistema de conexión a bases de datos y la escasez de frameworks de desarrollo web que existen, descartan estos lenguajes a la hora de desarrollar aplicaciones Web.

Java a diferencia de otros lenguajes, tiene un manejo muy bueno de conexiones a base de datos, la portabilidad entre sistemas operativos es muy buena ya que el código Java es ejecutado en su propia maquina virtual, lo que no la hace dependiente de librerías propias de un sistema operativo y tiene una serie de frameworks de desarrollo de aplicaciones Web que hacen que este lenguaje sea altamente productivo a la hora de desarrollar aplicaciones Web. Todos estos factores decantan al lenguaje de programación Java como la mejor opción para desarrollar aplicaciones Web.

#### *2.4.3.2 Ruby como elección final*

Dado que es un proyecto de final de carrera, y uno de los objetivos de éste es reforzar los conocimientos adquiridos durante esta. Se ha tenido en cuenta los conocimientos previos en algún lenguaje de programación. En este caso tan solo se tenía nociones de programación no orientada a objetos. Por lo que el aprendizaje de Java debía realizarse prácticamente desde cero.

Java es un lenguaje de programación, del cual su curva de aprendizaje puede llegar a ser bastante compleja. Tras la consulta a diversos expertos del sector, se ha planteado la opción de aprender y desarrollar este proyecto en un lenguaje más sencillo de aprender y cuyo conocimiento sirva de base para la posterior formación en Java.

Con dicho propósito se ha escogido como lenguaje de programación del presente trabajo de final de carrera el lenguaje RUBY [25].

#### *2.4.3.3 Ruby*

Lenguaje de programación creado recientemente, cuya primera versión fue liberada en 1995 por Yukihiro Matsumoto (Matz). Se trata de un lenguaje enfocado a la simplicidad y a la productividad, lo que favorece un desarrollo rápido y sencillo de aplicaciones. Su sintaxis elegante y natural, genera un código que se asemeja mucho mas al lenguaje humano que al computacional y como consecuencia es un código fácil de leer y entender.

Cada día este lenguaje va ganando más adeptos, tanto así que la empresa Sun Mirosystems, está apoyando un proyecto llamado Jruby que es un intérprete de Ruby escrito 100% en Java.

Entre las características más importantes de Ruby se pueden encontrar características similares con Java las cuales corroboran que el aprendizaje de Ruby puede ser una buena base para la posterior formación en Java.

Comunes Ruby - Java:

**Orientado a Objetos**: Ruby es un lenguaje totalmente orientado a objetos es decir que sigue estrictamente el paradigma de la programación orientada a objetos (POO) . En la POO todo se basa en entidades de la vida real con las que podemos interactuar, estas entidades son llamadas Objetos y son tres las características principales que las definen: Identidad, Comportamiento y Estado.

La programación Orientada a Objetos implica el uso de términos como: Clases, Objetos, Instancias, Atributos, Interfaces, Variables/métodos de Instancia, métodos/variables de clase, herencia, etc.

Java también es un lenguaje orientado a objetos por lo que la terminología usada en la programación de Ruby y Java es la misma.

**Herramientas de creación de tareas y gestión de dependencias**: Ruby dispone de una herramienta llamada Rake [26] que permite la creación de tareas repetitivas y la gestión de dependencias de las tareas de mantenimiento.

En Java existe una herramienta bastante similar llamada Ant [27] que permite la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción (build).

**Multiplataforma**: Ruby dispone de dos implementaciones, el interprete oficial de Ruby, que es el más utilizado y el interprete JRuby el cual es una implementación realizada en Java que funciona en una Maquina Virtual Java. Ambas implementaciones del intérprete las podemos encontrar en multitud de sistemas operativos como: sistemas Linux, Mac OS X, Microsoft Windows, Windows CE y la mayoria de Unix. Además desde la versión 1.9 Ruby ha sido portado a Symbian OS 9.x.

**Entornos de desarrollo Web (Frameworks)**: Ruby dispone de diversos Frameworks para el desarrollo de aplicaciones Web. El más famoso y más completo es Rails que junto con Ruby forman la plataforma de desarrollo Ruby on Rails, la cual ha sido la escogida para realizar este proyecto.

Java también dispone de múltiples Frameworks de desarrollo Web que ayudan a programar siguiendo el patrón de diseño MVC, Struts, Spring, Hibernate

**Software Libre**: Tanto Ruby como Java disponen de una licencia GPL, del inglés GNU General Publical License, creada por la Free Software Foundation la cual hace que estos lenguajes de programación sean software libre y por tanto los usuarios tienen total libertad para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software;

**Gestion Automatica de memoria**: Ruby dispone de un “recolector de basura” que gestiona automáticamente la reserva de memoria del ordenador. Gracias a la gestión automática de memoria no se ha de invocar ninguna subrutina para liberar espacios de memoria ocupados, ya que es el propio intérprete de Ruby el que gestiona el ciclo de vida de los objetos creados.

Java al igual que Ruby resuelve el problema de fugas de memoria con un recolector de basura automatico que es gestionado por el Runtime de Java.

**Generador de documentación**: Ruby dispone de un generador de código por defecto llamado Rdoc [28]. Él generador parsea los archivos .rb, .rbw y .c del proyecto y busca los comentarios situados encima de las definiciones de las funciones. Con estos datos se crean archivos HTML[xii] formando una fuente de documentación de librerías muy útil a la hora de reutilizar código.

Java también dispone de un generador similiar a Rdoc. La herramienta Java para la creación de documentación embebida es Javadoc.

No comunes Ruby - Java:

**Lenguaje Interpretado**: Al ser un lenguaje totalmente interpretado [29], no requiere ser compilado antes de ejecutarse, ya que la ejecución del código se realiza a través de un intérprete que crea una representación intermedia del código que se va compilando a medida que va a ser ejecutado.

Java a diferencia de Ruby si que necesita ser compilado, ya que aunque comúnmente el código también es transformado a una representación intermedia (bytecode) antes de ser compilado a código máquina, toda la compilación se realiza de una vez.

**Dinamismo**: En Ruby, las clases nunca están cerrada esto quiere decir que siempre se pueden añadir nuevos métodos a esta clase. Esto es válido tanto para las clases incluidas con el intérprete como para las nuevas clases escritas.

**Tipado Dinámico y 'Duck Typing'**: Ruby es un lenguaje de tipos débiles, es decir que las variables que usadas no tienen un tipo fijo y no es necesaria su declaración para poder usarlas. En Ruby, se confía menos en el tipo de un objeto y más en sus capacidades que significa que el tipo de un objeto está definido por lo que puede hacer, no por lo que es. En la comunidad Ruby se dice que 'Si un objeto camina como un pato (*walks like a duck*) y habla como pato (*talks like a duck*), entonces el intérprete es feliz de tratarlo como si fuera un pato'.

El tipado dinámico aporta flexibilidad a la hora de programar, ya que libera al programador de la obligación de tener que declarar todas las variables usadas en el desarrollo del código. Esto conlleva la creación de programa con menor número de líneas y en consecuencia un aumento de productividad. Pero se puede convertir en una desventaja, ya que el uso del tipado dinámico puede comportar la generación de pequeños bugs difícilmente detectables hasta la ejecución del programa. Por esta razón el desarrollo de aplicaciones en Ruby requieren que se creen multitud de test para poder detectar el correcto funcionamiento del código generado.

Java contrariamente a Ruby es un lenguaje de programación de tipado fuerte, es decir que sí se han de declarar e inicializar las variables.

**Sintaxis Concisa, compacta**: La Sintaxis de Ruby permite escribir bloques de código muy compactos y en un lenguaje muy "humano". En consecuencia permite crear programas con

[xii] HyperText Markup Language (*Lenguaje de Marcas de Hipertexto*)

menor número de líneas de código que se refleja en una menor cantidad de puntos de posibles errores de tipado.

La sintaxis de Java es menos compacta y más “computacional”, lo que la hace menos legible. En el siguiente ejemplo se puede ver claramente las diferencias entre ambos lenguajes:

```
En Java:

for(int i=0;i<3;i++) { System.out.print("titulo ejemplo EspiroQ");}

En Ruby:

3.times do print "titulo ejemplo EspiroQ " end
```

### 2.4.4 Entorno de desarrollo Web

La definición de Entorno de trabajo Web (en inglés Framework) es un debate que todavía está abierto a discusión, pero se podría decir que esencialmente, un Framework, es conjunto de aplicaciones y utilidades de soporte y código que simplifican las tareas de los programadores.

Actualmente se pueden encontrar una gran variedad de entornos de desarrollo Web que están basados en Ruby, por lo que determinar cuál es el más adecuado para el presente proyecto puede resultar complicado. Por esta razón se debe tener en cuenta una serie de factores para garantizar una correcta valoración de las prestaciones ofrecidas por cada uno ellos.

Tal y como se menciona en la sección 2.4.3.2 de éste documento, uno de los objetivos de la realización del Trabajo de Final de Carrera es el de adquirir y reforzar nuevos conocimientos, en este caso acerca de la programación de aplicaciones Web. Por lo que uno de los factores a tener en cuenta en la elección del Framework de desarrollo es que aparte de facilitar las tareas repetitivas, también ha de ayudar a programar siguiendo “buenas prácticas” de programación.

La filosofía de diseño MVC (Modelo-Vista-Controlador) es una de éstas buenas prácticas. Este patrón de diseño se utiliza para separar la lógica de negocio de la representación de la información en la interfaz de usuario. Sin entrar en profundidad en el análisis de éste patrón ya que es analizado posteriormente (2.5.2.1), al seguir el patrón MVC se facilita la modificación de la lógica de negocio o de la interfaz de usuario, sin que una afecte a la otra, con lo cual se incrementa la flexibilidad y la reutilización.

Dentro de este patrón se debe diferenciar entre dos arquitecturas:

• Arquitectura Push-based.

También se conoce como server push. Se caracteriza en que la petición para que se sirva una determinada información se produce en el servidor. Esto implica que el servidor se encarga de inyectar los contenidos hacia el cliente, sin que se produzca la petición previa.

• Arquitectura Pull-based.

En esta tecnología, las peticiones de información se originan en lado cliente y el servidor sólo se encarga de enviar la respuesta a estas peticiones.

Siguiendo con los factores a analizar para la elección final del entorno de desarrollo se han estudiado las siguientes características de los Frameworks de desarrollo Web:

- Soporte para Ajax.

  Se proporcionan los medios necesarios para mejorar la interactividad, velocidad y usabilidad de los sitios Web.

- MVC.

  Se facilita la separación de la lógica de negocio de la interfaz de usuario mediante capas siguiendo el patrón de diseño Modelo – Vista - Controlador.

- Internacionalización y localización.

  Se ofrece el soporte para poder adaptarse a diferentes idiomas y regiones sin necesidad de cambios de ingeniería, ni cambios en el código.

- Interacción con bases de datos.

  Se incorporan APIs para poder trabajar a alto nivel con diferentes tipos de bases de datos sin necesidad de modificar el código. Adicionalmente se pueden suministrar herramientas de mapeo entre objetos y estructuras relacionales (ORM), de gestión transaccional y de migración de bases de datos.

- Testing.

  Se provee de funcionalidades para realizar todo tipo de testing.

- Seguridad.

  Se provee de funcionalidades para gestionar la autenticación y autorización a los usuarios en base a determinados criterios.

- Uso de plantillas.

  Mediante el uso de plantillas se puede generar contenido de forma dinámica reduciendo drásticamente el número de páginas Web del sitio.

- Uso de caché.

  En algunos entornos se utiliza la caché para almacenar temporalmente documentos con el fin de reducir la carga del servidor y el tiempo de respuesta.

- Validación de formularios.

  Se suministran las herramientas necesarias para realizar la validación de la correcta entrada de datos en formularios Web.

En la siguiente tabla se puede ver una comparativa de los entornos de desarrollo web de Ruby [30].

| Framework | Ajax | MVC | MVC Push/Pull | Internacionalización y localización | ORM | Testing | Seguridad | Uso de plantillas | Uso de caché | Validación de formularios |
|---|---|---|---|---|---|---|---|---|---|---|
| **Camping** | No | Si | Push | No | Patron Active record | via Mosquito | No | Si | No | No |
| **Nitro** | jQuery | Si | Push | Si | Og | RSpec | Si | Si | Si | Si |
| **Ruby on Rails** | Prototype, script.aculo.us | ActiveRecord Action Pack | Push | Localization Plug-in | ActiveRecord | Unit Tests, Functional Tests and Integration Tests | Plug-in | Si | Si | Si |
| **Sinatra** | No | Si | Push | No | ORM-independent | Rack::test | No | Si | A traves del middleware de Rack | No |

**Tabla 2 - Comparativa Entornos desarrollo Web - Ruby**

#### 2.4.4.1 Rails como elección final

Finalmente se ha decidido utilizar Rails [31] como entorno de desarrollo web. Esta decisión se fundamenta en varios motivos. Para empezar Ruby on Rails es actualmente el Framework más completo de todos los analizados anteriormente, lo que lo perfila como una herramienta de grandes prestaciones, tanto en productividad, como en flexibilidad y facilidad de mantenimiento.

Otro aspecto a destacar es que Rails es sin duda el Framework más popular dentro de la comunidad de programadores de Ruby, lo que le aporta un apoyo absoluto por parte de desarrolladores, lo que tiene su efecto en un crecimiento exponencial de la comunidad rails (Foros, Lista de correos, Screencast, Blogs, Libros, Documentación online, etc)

Finalmente tras estudiar brevemente la demanda de la industria del software (Ilustración 11) muestra la gran apuesta del sector por este entorno. Estas circunstancias erigen a Ruby on Rails como un entorno de desarrollo con grandes perspectivas de futuro dentro del panorama de aplicaciones Web.

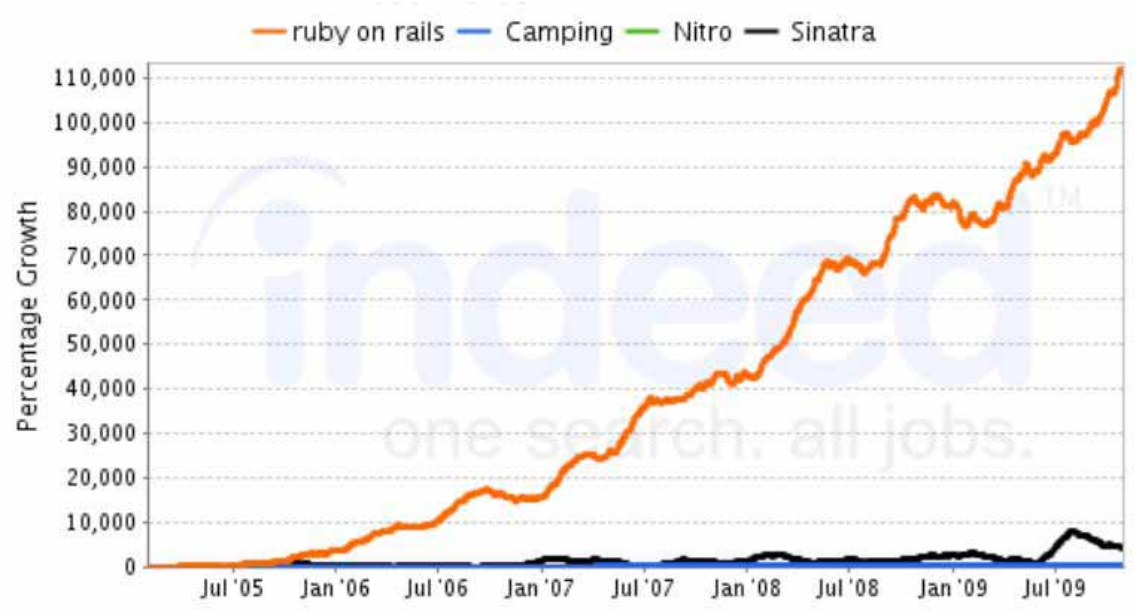



Ilustración 13 - Tendencia del crecimiento de la demanda Framework Ruby. - Indeed.com

### 2.4.5 Base de Datos

En un primer momento se ha sondeado la posibilidad de utilizar una base de datos Oracle, ya que existen varios motivos que motivan su uso. El primero de estos motivos es que interesa migrar los datos generados en el estudio E-Spir@p a la nueva aplicación EspiroQ y el segundo motivo es la experiencia anterior con la aplicación E-Spiro que ha demostrado que su elección aportaría muchas ventajas como pueden ser las de fiabilidad, robustez, gran numero de expertos en Oracle, etc.

Pero el primer factor de decisión ha sido el tipo de licencia de la Base de Datos que ha desechado inmediatamente la posibilidad de utilizar Oracle.

Dentro del mercado de Bases de Datos con licencia GPL, se ha escogido MySQL [32].

A parte de que MySQL es muy popular por el tipo de licencia y por la velocidad de procesamiento la decisión de seleccionar ésta base de datos se ha tomado porque forma parte del conjunto de herramientas instaladas en el servidor de proyectos europeos del Hospital Clínic de Barcelona, que es el lugar donde se ha de poner en producción la aplicación.

### 2.4.6 Servidor contenido dinámico

Los servidores Web Ruby mas conocidos son Webrick, Mongrel, Thin y Passenger. Por razones de facilidad de instalación y de volumen de documentación se ha decidido escoger uno de los los dos servidores que se instalan junto la última versión de Rails (2.3.5) Webrick y Rails

Entre Webrick y Mongrel se ha escogido Mongrel [33]. Mongrel es un servidor que gestiona mucho más velozmente que Webrick las peticiones web. Aunque tiene el mismo modelo de concurrencia que Webrick el modelo de Mongrel se ha optimizado gestionando así mucho mejor los multi hilos. Finalmente decir que Mongrel es un muy buen servidor para soportar cargas moderadas de carga que por las características de este proyecto son más que suficientes.

### 2.4.7 Servidor contenido estático

Al igual que la elección de la base de datos (2.4.5), la configuración previa del servidor de producción impone que el servidor de contenido web de salida de la red del Hospital Clínic ha de ser un servidor Apache 5.x.

La primera idea ha sido usar Apache mas Fast CGI para ejecutar el contenido estático del proyecto, pero tras realizar diversas consultas en internet se ha visto que el rendimiento final no era demasiado bueno.

Al final se ha decidido utilizar Apache tan solo como proxy, y utilizar Mongrel como servidor de contenido dinámico y estático.

Entonces en el entorno de desarrollo se usa Mongrel como servidor de contenido estatico y en el entorno de producción se utiliza Mongrel como servidor Web y Apache como proxy de las peticiones http/https hacia Mongrel.

### 2.4.8 Sistema de control de versiones

Un sistema de control de versiones, en inglés CVS, es una aplicación que ayuda a automatizar las tareas de guardar, recuperar y registrar todo cambio realizado sobre un fichero. Es una herramienta imprescindible para controlar la evolución de un proyecto y poder gestionar todos los cambios realizados en él.

Actualmente el sistema de versiones GIT se esta alzando como preferencia dentro de la comunidad de Ruby on Rails, aunque Subversión sigue teniendo mucha popularidad. En este proyecto se ha decidido optar por Subversion [34] usando TortoiseSVN [35] como interfaz gráfica.

Subversion es un software de sistema de control de versiones de código abierto y gratuito el cual maneja ficheros y directorios a través del tiempo. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios.

## 2.5 Patrones

El uso de patrones de diseño nace de la necesidad de mejorar la calidad del software partiendo de la experiencia previa existente.

Para el correcto uso de patrones es necesario detectar bien el problema y saber encontrar el patrón que se adapta.

### 2.5.1 Principios de diseño

Los principios de diseño ayudan a evaluar la calidad del diseño del sistema. Así se podrá decir que un diseño tiene una buena calidad si se ajusta a los principios de diseño.

Aquí se explican brevemente tres de ellos, en los que se ha tenido especial cuidado durante el diseño de este proyecto.

#### *2.5.1.1 Bajo acoplamiento*

El acoplamiento es la medida de dependencia entre elementos del sistema (por ejemplo clases). Interesa que el acoplamiento sea bajo para mantener el código sencillo de leer e interpretar. Si dos elementos se ven acoplados un cambio en uno de ellos puede cambiar todo el sistema, es más difícil depurar el código y reutilizar las clases se complica.

#### *2.5.1.2 Alta cohesión*

La cohesión es la medida de la relación que hay entre las responsabilidades de una misma clase. La idea es que una clase tenga un ámbito de trabajo específico y sólo se encargue de él. Igual que el bajo acoplamiento, tener una alta cohesión ayudará a entender el código, reutilizar las clases y facilitará el mantenimiento.

#### *2.5.1.3 No repetición*

Como su nombre indica, se ha de procurar que no haya responsabilidades ni informaciones repetidas en el código. Este principio ayuda especialmente para corregir errores, ya que una responsabilidad sólo se ha de encontrar en un sitio.

### 2.5.2 Patrones de arquitectura

Son los que se usan al definir la arquitectura del sistema, así que hay que tener cuidado puesto que las decisiones tomadas aquí afectarán a todo el diseño y desarrollo.

Una de las ventajas de utilizar el framework Ruby on Rails como plataforma de desarrollo es el uso inteligente que hace de patrones de diseño conocidos. Y en consecuencia la necesidad de seguir buenas prácticas de programación.

#### *2.5.2.1 Modelo, vista y controlador*

En la mayoría de entornos de desarrollo de aplicaciones Web se sigue la filosofía de diseño MVC (Model view controller). Este patrón de diseño se utiliza para separar la lógica de negocio y la lógica de control de la representación de la información en la interfaz de usuario. Mediante esta práctica se consigue la independencia entre capas, haciendo que un cambio en una de ellas no afecte a las demás. Con lo cual se incrementa la flexibilidad y la reutilización. Por ejemplo, en un momento dado se requiere un cambio de tecnología para la presentación no hace falta modificar todo el proyecto, únicamente la vista.

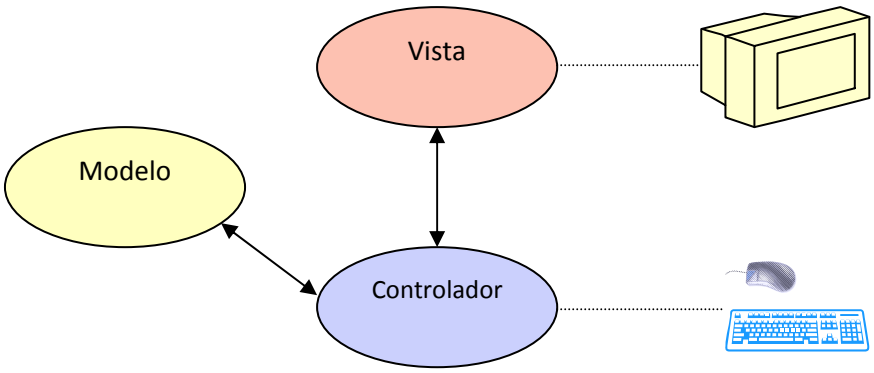



Ilustración 14 - Arquitectura MVC

Separación en tres capas:

- Modelo: lugar dónde se implementa la lógica de negocio, encapsula el estado del sistema y se limita a la vista y su controlador, facilita las presentaciones complejas. Notifica a la vista los cambios en el estado del sistema.
  En el código dentro del package "/app/models/" se pueden encontrar las clases del modelo.

- Vista: presenta los datos al usuario y recoge sus peticiones para ser enviadas posteriormente al controlador.
  El package "/app/views/" es el lugar donde se encuentran las vistas.

- Controlador: responde a las acciones del usuario y las relaciona con los acontecimientos del sistema. Decide qué vista debe mostrarse al usuario e invoca cambios en el modelo.
  En el proyecto se encontraran en el package "/app/controllers/", que escucha las peticiones de la vista y las transmite al modelo, luego devuelve las respuestas.

Aplicando el patrón modelo-vista-controlador se consigue alta cohesión, ya que las responsabilidades están delimitadas. Y en consecuencia es posible disponer de un equipo de desarrolladores especializados en áreas de trabajo diferentes, los que pueden hacer modificaciones en cualquier capa sin afectar a las otras.

#### *2.5.2.2 Rails y Modelo Vista Controlador (MVC)*

Rails es un Framework MVC, esto quiere decir que el mismo entorno de desarrollo fuerza a programar a siguiendo estrictamente este patrón de diseño.

En Rails la gestión de las clases de cada capa del patrón MVC es gestionada por un modulo diferente: Active Record gestiona el Modelo, Action Controller gestiona los Controladores y finalmente Action View gestiona las vistas.

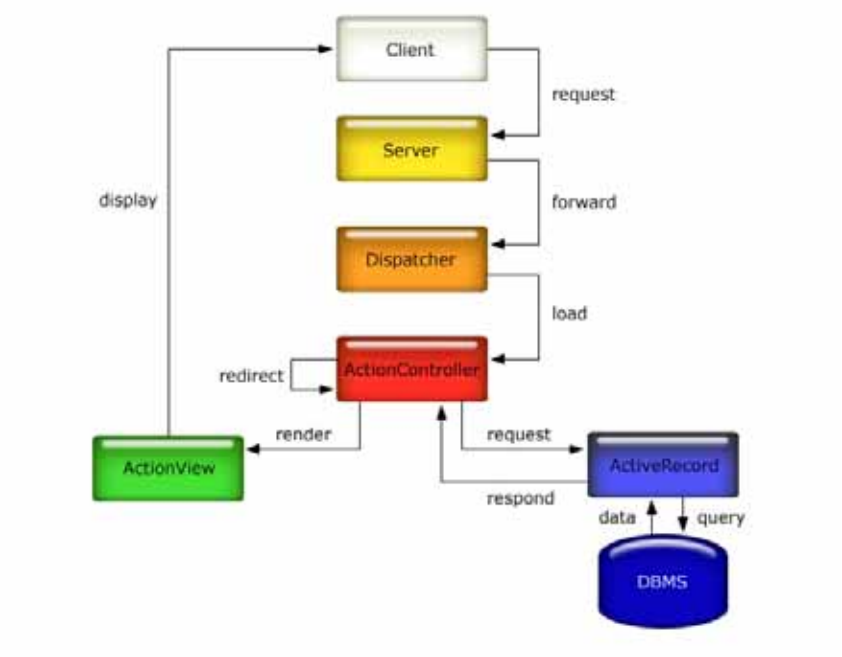



Ilustración 15 – Arquitectura MVC en Rails

### 2.5.3 Patrones de diseño

Se utilizan para solucionar problemas concretos, que no afectan a todo el sistema, sólo a partes especificas.

La programación de aplicaciones a través del entorno Ruby on Rails conlleva la utilización de patrones de diseño. En la realización de éste proyecto se pueden detectar el uso de los siguientes patrones:

#### *2.5.3.1 ActiveRecord (AR)*

Active Record es un punto de vista de la forma en que se accede a los datos de la base de datos. Un objeto representa una fila de la tabla de datos, encapsulando el acceso a la base de datos y añadiéndole lógica de negocio.

Citando al creador del patrón,Martin Fowler, (“Patterns of Enterprise Architecture”) ActiveRecord [36] es:

*“Un objeto que engloba una fila de una tabla o vista de la base de datos, encapsula el acceso a la base de datos, y agrega lógica del dominio del problema sobre estos datos.”*

Esto quiere decir que gran parte de este patrón tiene su origen en el Modelo de Dominio, lo que significa que las clases están muy cercanas a su representación en la base de datos. Cada objeto Active Record es responsable de si mismo tanto en la persistencia como en la lógica de negocio. Es decir cuando se crea uno de estos objetos, se añade una fila a la tabla de la base de datos. Cuando se modifican los atributos del objeto, se actualiza la fila de la base de datos.

```
def index
  @center = current_user.professional.center
end
def show
  @center = Center.find(params[:id])
  currentCenter = current_user.professional.center
```

### 2.5.3.2 *Iterador (Iterator)*

Creado para evitar exponer la estructura interna de los datos almacenados. La clase Iterador deberá conocer la estructura y el modo de recorrerla.

En el código del presente proyecto se puede encontrar el uso del patrón iterador en múltiples ocasiones. Un ejemplo se puede encontrar en la clase Center del modelo.

```
def getAllNumSpirosCertificated
  numspiros = 0
  numspiros = self.getNumSpirosCertificated
  if self.morechildren?
    self.getchildren.each do |c|
      numspiros += c.getAllNumSpirosCertificated
    end
  end
  return numspiros
end
```

Ilustración 16 - codigo EspiroQ - patron iterador

### 2.5.3.3 *Factoria simple (Simple Factory)*

Simple Factory retorna una instancia de un conjunto de posibles clases dependiendo de los datos que han sido pasados.

En Ruby todas las clases son Objetos por lo que este patrón es utilizado implícitamente

### 2.5.3.4 *Otros*

A parte de los patrones de diseño nombrados anteriormente, el entorno de desarrollo Ruby on Rails tiene implementados distintos patrones como pueden ser: Singleton, Observer, Command, los cuales pueden ser instanciados en el momento que se desee.

## 2.6 Documentos de diseño

### 2.6.1 Funcionalidades

En este apartado se describen a grandes rasgos los requisitos funcionales que ha de cumplir la aplicación a desarrollar en el presente proyecto.

Funcionalidades generales

- Listado filtrado de pacientes sobre los que se está trabajando
- Selección del paciente a gestionar

- Listado de espirometrías de un paciente
- Listado de espirometrías de un centro
- Listado de espirometrías realizadas por un profesional
- Detalle de la espirometría seleccionada:
- Representación gráfica de la medición (volumen-tiempo y flujo volumen)
- Representación tabular de los datos relevantes
- Información sobre la calidad de la misma introducida por el grupo experto
- El grupo experto utiliza la misma funcionalidad, a la que se añade la opción de puntuar una espirometría en concreto mediante una serie de calificadores definidos y delimitados.

Funcionalidades de gestión:

- Fijar los objetivos de los centros, profesionales y evaluar on-line el cumplimiento de dichos objetivos.
- Consulta de estadísticas operativas
- Consulta de estadísticas sanitarias
- Fijar Objetivos a los profesionales

Equipamiento:

El aplicativo inicialmente soporta los siguientes espirómetros:

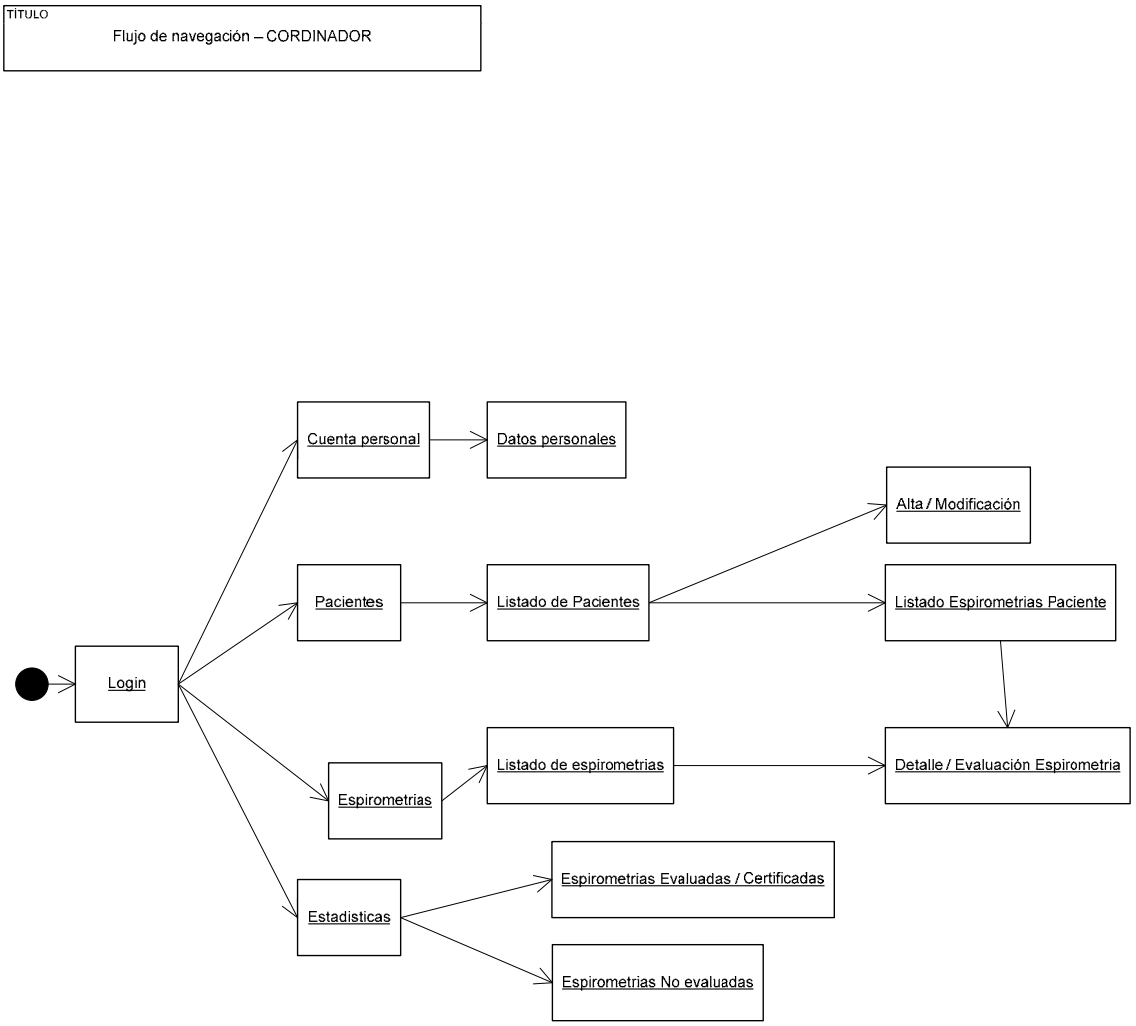
- NDD EasyOne

Pero se ha de preparar para soportar todos aquellos equipos capaces de generar ficheros en formato XML

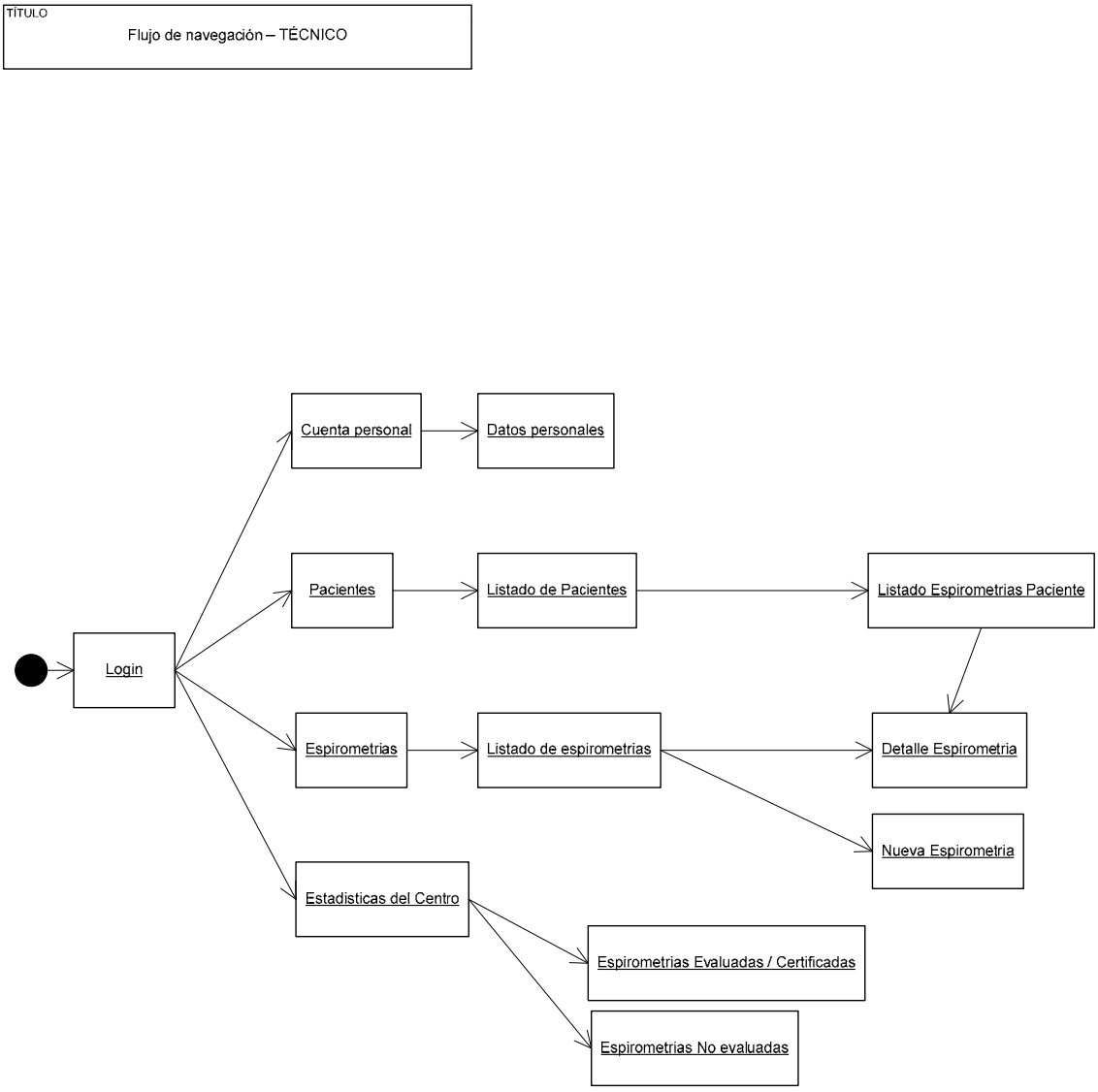
### 2.6.2 Flujo de navegación

El flujo de navegación muestra gráficamente los posibles flujos de navegación por la aplicación según el rol de usuario.

### 2.6.2.1 Coordinador



### 2.6.2.2 Técnico




## 2.6.3 Modelos

El paso previo al diseño del modelo de datos de la aplicación ha sido el de realizar un estudio de la aplicación E-Spiro.

Con la intención de integrar los datos generados por la nueva aplicación a desarrollar con los datos ya generados en el estudio e-Spir@p se ha estudiado la posibilidad de aprovechar el antiguo modelo de base de datos. Pero tras realizar un análisis general se ha llegado a la conclusión de que el modelo de datos de la aplicación E-Spiro no es aprovechable para el presente proyecto. Se ha determinado que el modelo de datos de E-Spiro es poco escalable y presenta algunas deficiencias de diseño por lo que se ha tomado la decisión de realizar un nuevo modelo con un nuevo diseño de las tablas que permita desarrollar una aplicación escalable, de fácil mantenimiento y integrable con las nueva plataforma Linkcare que está siendo desarrollada en el Hospital Clínic de Barcelona.

### 2.6.3.1 Modelo Base de datos de E-spiro

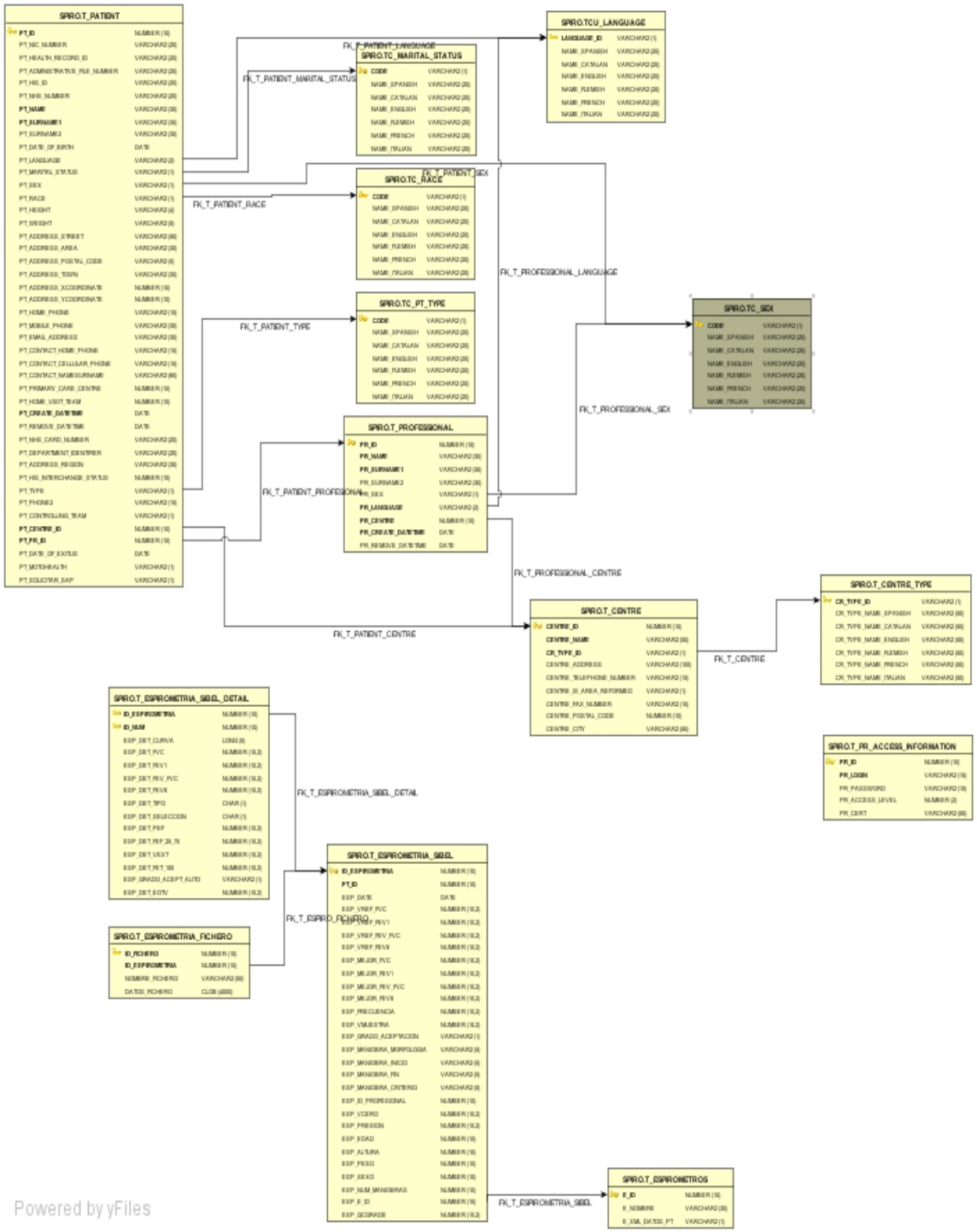



Ilustración 17 - Modelo Base de Datos E-Spiro

El modelo de datos de la aplicación E-Spiro no permite cumplir los requisitos funcionales definidos en éste proyecto. Entre otras limitaciones de estructura se ha encontrado que:

**Datos de paciente dentro de tabla paciente**: Este diseño implica que un paciente no pueda disponer de más de una dirección/teléfono de contacto.

**Gestión de Centros**: No existe la posibilidad de crear agrupaciones de centros por lo que no permite gestionar zonas sanitarias.

**Validación Espirometrías**: Debido al modelo de datos solo es posible revisar la espirometría una sola vez. Y no es posible que se realice la revisión por más de un profesional.

### 2.6.3.2 Modelo de datos - EspiroQ

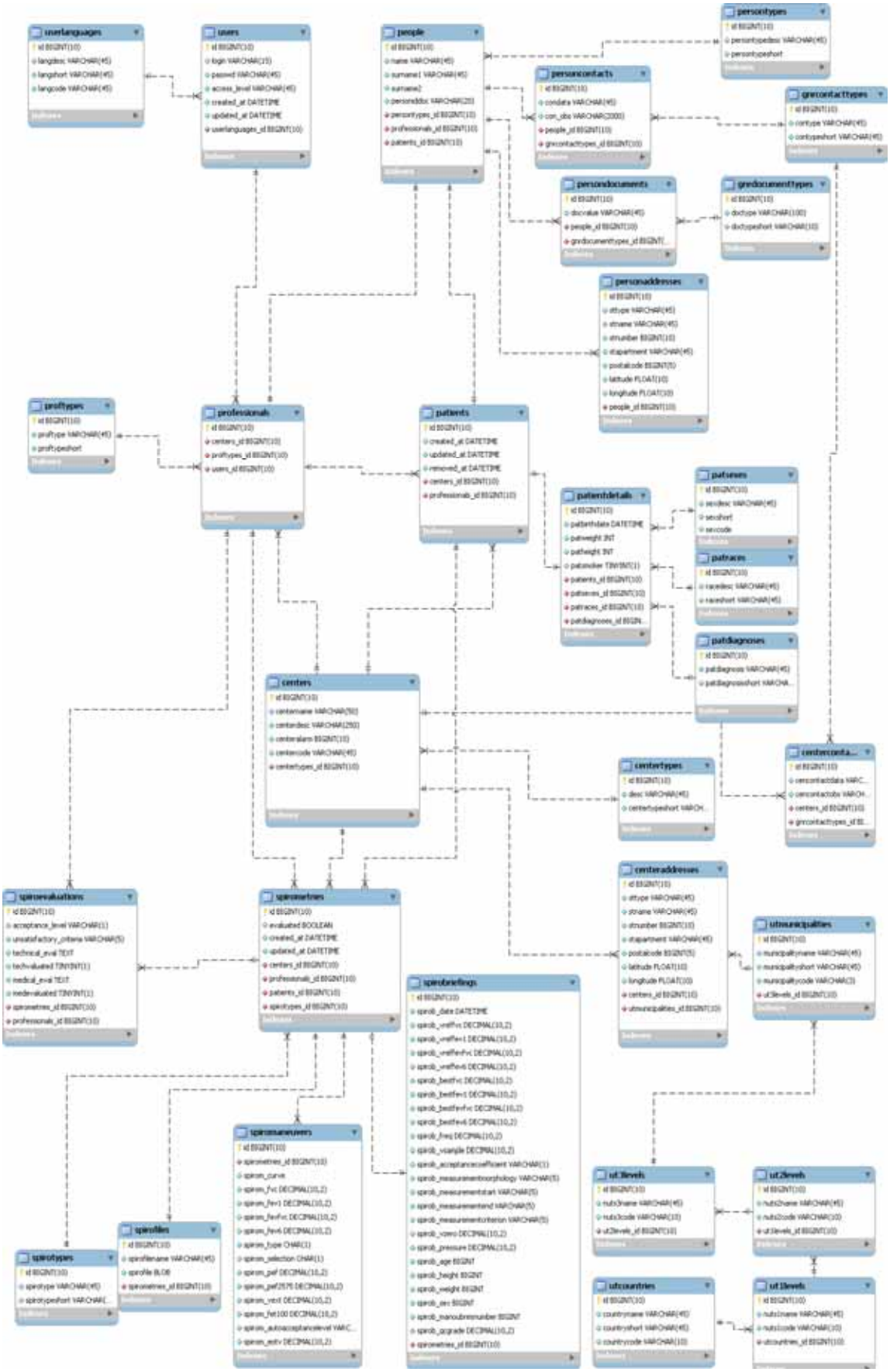


Ilustración 18 - Modelo Base de Datos EspiroQ

El nuevo modelo de datos diseñado ha tenido en cuenta requisitos básicos en el diseño de bases de datos como los de tener una estructura normalizada de las tablas, controlando la redundancia de la información, manteniendo la consistencia de datos, evitando perdidas de información y manteniendo la integridad de los datos guardados. También se ha optimizado el diseño para tener el acceso más eficiente posible a la información.

La elección de los nombres de las tablas se ha efectuado siguiendo las convenciones de Ruby on Rails, donde el Active Record obliga a que las tablas de la base de datos tengan el mismo nombre que las clases del modelo pluralizadas.

Finalmente se ha tenido en cuenta el requisito (2.3.9) que dice que en un futuro la aplicación de control de calidad de espirometrías se ha de integrar con la plataforma de telemedicina Linkcare. Por esta razón las principales entidades del modelo de datos tienen la misma estructura que las entidades del modelo de datos de Linkcare.

Las entidades que se han diseñado con la misma estructura son: *spirometries, spiromaneuvers y spirofiles*, y con una estructura muy similar encontramos *users, professionals, patients* y *centres*. Gracias a las separación entre capas seguida en el diseño de la aplicación, para realizar la integración entre aplicaciones no se ha de modificar código fuente, tan solo se ha de modificar las configuraciones referentes al motor de la base de datos utilizado en la persistencia de los objetos.

El modelo de datos consta de una serie de tablas principales que representan las entidades de la aplicación. Éstas definen la lógica del modelo, y son apoyadas por una serie de tablas auxiliares de definición de datos.

A continuación se enumeran las tablas en grupos funcionales

**Espirometrías**: La entidad (tabla principal) es *spirometries* y es complementada por las tablas *spirobriefings, spiroevaluations, spirofiles, spiromaneuvers y spirotypes*.

Spirometries: La tabla *Spirometries* representa la entidad Espirometría. Una espirometría pertenece a un centro, un profesional y un paciente, ademas de a un tipo de espirómetro. A su vez una espirometría tiene una o más maniobras (*spiromaneuvers*), unos detalles (*spirobriefings*) de la espirometría, una o más evaluaciones (*spiroevaluations*) y un archivo (*spirofiles*) con los datos xml

**Personas**: Se gestiona los usuarios según el rol dentro de la aplicación, profesional o paciente. Por esta razón hay 2 entidades*, professionals y patients* que comparten las tablas complementarias que las definen como personas. Estas tablas complementarias son *gnrcontacttypes, gnrdocumenttypes, patdiagnoses, patientdetails, patraces, patsexes, people, personaddresses, personcontacts, persondocuments, persontypes, proftypes y users.*

*Professionals :* La entidad Profesional *(professionals)* pertenece a un usuario (*users)*, un tipo de profesional (*proftypes*) y a un centro (*centres*).

*Patients*: La entidad Paciente (*patients*) contiene la información básica de identificación del paciente (nombre, apellido1, apellido 2 y fechas de creación actualización y baja del paciente), además pertenece a un Profesional (*professionals*) y a un Centro (*centres*).

*People*: Tanto un profesional como un paciente son una persona (*people*) por lo que la entidad Persona (*people)* está enlazada con un Profesional (*professional*), con un Paciente (*patients*) y en previsión de añadir nuevos tipos de roles en un futuro también está enlazado con un tipo de persona (*persontypes)*.

**Centros**: La tabla principal que define la entidad Centro es *centers* y es complementada por las tablas *centeraddresses, centercontacts y centertypes, ut1levels, ut2levels, ut3levels, utcountries, utmunicipalities.* Hay que destacar que en las tablas *utxxx* se guarda la persistencia de la localización geográfica de los centros en un formato integrable con cualquier base de datos que siga la nomenclatura europea de representación de datos de geolocalización.

# 3 DESARROLLO

## 3.1 Programación

El desarrollo de la aplicación ha supuesto programar 1774 líneas de código (LOC, Lines Of Code):

| Name | Lines | LOC | Classes | Methods | M/C | LOC/M |
|---|---|---|---|---|---|---|
| Controllers | 671 | 405 | 9 | 28 | 3 | 12 |
| Helpers | 108 | 48 | 0 | 4 | 0 | 10 |
| Models | 1442 | 794 | 29 | 112 | 3 | 5 |
| Libraries | 220 | 111 | 0 | 23 | 0 | 2 |
| Integration tests | 0 | 0 | 0 | 0 | 0 | 0 |
| Functional tests | 183 | 141 | 9 | 19 | 2 | 5 |
| Unit tests | 355 | 275 | 35 | 14 | 0 | 17 |
| Total | 2979 | 1774 | 82 | 200 | 2 | 6 |

Tabla 3 - Resumen líneas de código EspiroQ

### 3.1.1 MVC

La aplicación ha sido desarrollada, tal como se había definido en las fases de diseño, siguiendo el patrón Modelo – Vista – Controlador (MVC)

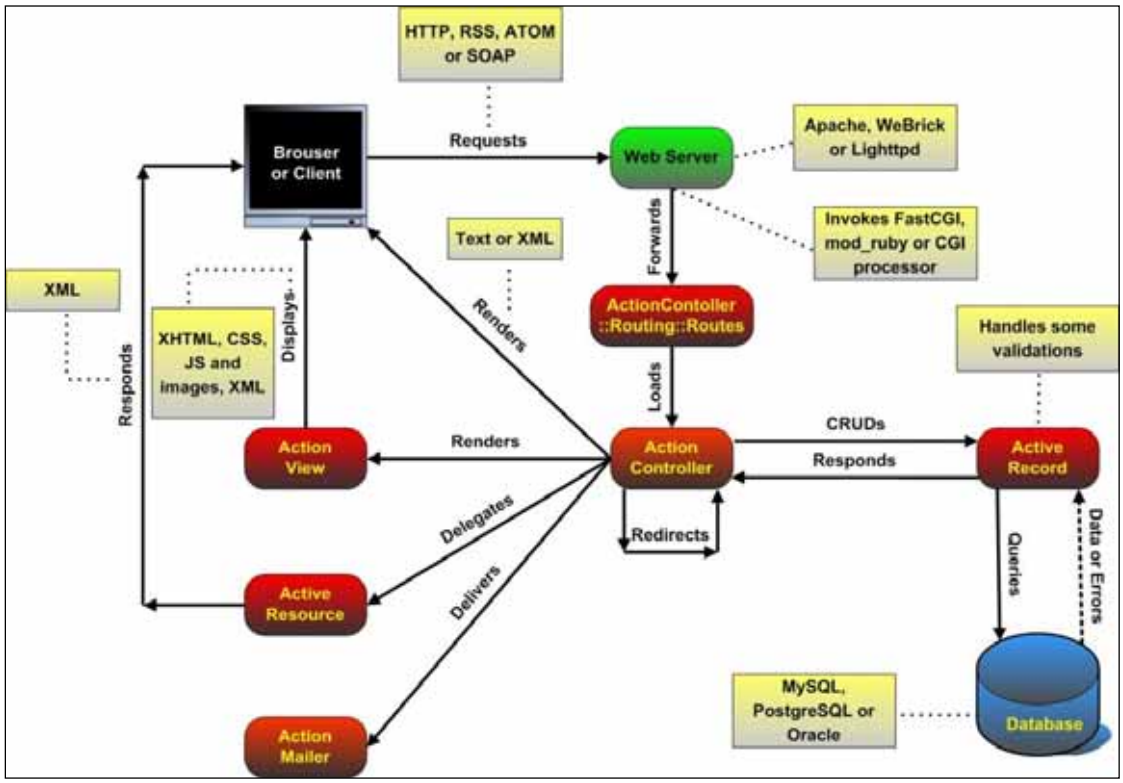



Ilustración 19 - modelo MVC - aplicación

El desarrollo se ha realizado en 3 fases diferentes: en la primera se ha creado el modelo de datos, y se ha programado la lógica de negocio. En una segunda fase se han creado los controladores, se ha definido el parseo (routing) de las peticiones web a los métodos de la aplicación y se han desarrollado los algoritmos necesarios para pintar las gráficas. Finalmente se ha desarrollado la presentación de la aplicación, es decir se han creado las plantillas HTML y se han creado las hojas de estilo CSS.

### 3.1.2 Active Record (AR) Modelo

La lógica de negocio de la aplicación se ha programado siguiendo el patrón ActiveRecord [36].

*"Un objeto que engloba una fila de una tabla o vista de la base de datos, encapsula el acceso a la base de datos, y agrega lógica del dominio del problema sobre estos datos."*

*Martin Fowler*

Gracias la implementación de las librerías de Ruby on Rails de AR, la mayoría de tareas de definición y manipulación de la base de datos se ha realizado mediante programación Ruby. A pesar de poder llevar toda la gestión de datos a través de metaprogramación Ruby también se han utilizado sentencias SQL que han permitido realizar consultas mucho más rápidas que las que se podían generar a través de AR.

#### *3.1.2.1 Definición de datos*

La definición de la base de datos se ha realizado íntegramente utilizando migrations (Ver Anexo A.B). Las migraciones son clases Ruby que ActiveRecord se encarga de interpretar y traducir en las secuencias SQL que haga falta según el motor de base de datos con el que se está trabajando.

```
class CreateSpiromaneuvers < ActiveRecord::Migration
  def self.up
    create_table :spiromaneuvers do |t|
    #primary key 'id' se crea automaticamente
    #creamos CLAVES FORANEAS
     t.references :spirometry, :limit => 10, :null => false #
    #fin CLAVES FORANEAS

    t.integer "id_num",               :limit => 10,
    t.text    "spirom_curve"
    t.decimal "spirom_fvc",                          :precision => 10, :scale => 2
    t.decimal "spirom_fev1",                         :precision => 10, :scale => 2
    t.decimal "spirom_fev_fvc",                      :precision => 10, :scale => 2
    t.decimal "spirom_fev6",                         :precision => 10, :scale => 2
    t.string  "spirom_tipo",          :limit => 1
    t.string  "spirom_seleccion",     :limit => 1
    t.decimal "spirom_pef",                          :precision => 10, :scale => 2
    t.decimal "spirom_fef_25_75",                    :precision => 10, :scale => 2
    t.decimal "spirom_vext",                         :precision => 10, :scale => 2
    t.decimal "spirom_fet_100",                      :precision => 10, :scale => 2
    t.string  "spirom_grado_acept_auto", :limit => 1
    t.decimal "spirom_eotv",                         :precision => 10, :scale => 2

      t.timestamps
    end
  end

  def self.down
    drop_table :spiromaneuvers
  end
end
```

Ilustración 20- código - migration spiromaneuvers

Otra de las ventajas de utilizar migraciones, es que ha permitido llevar un control de versiones de cambios del schema de la base de datos, permitiendo volver atrás tras una modificación errónea. Manipulación de datos – AR

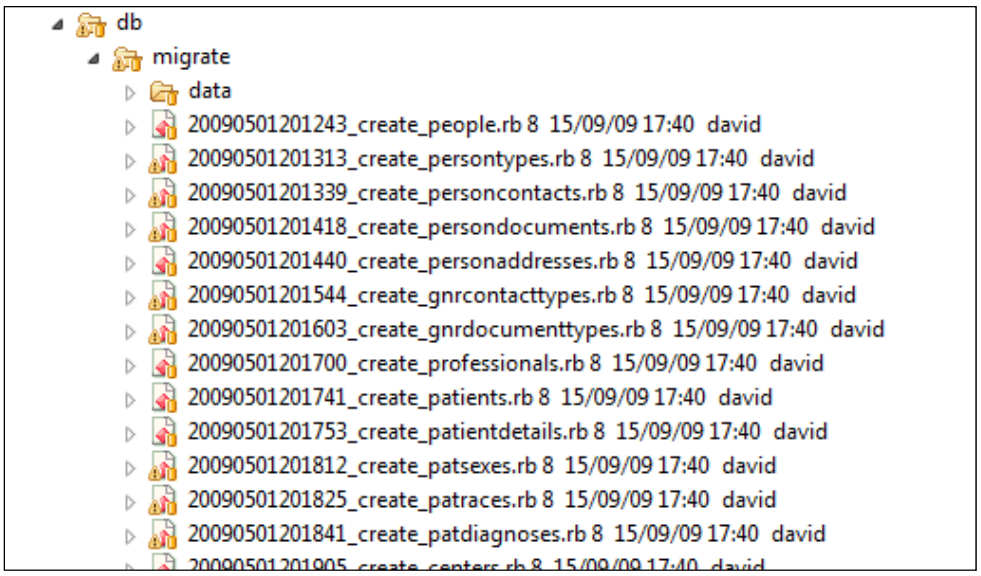



Ilustración 21 – control de versiones- migrations

### *3.1.2.2 Modelos y Lógica de negocio*

La API de ActiveRecord [37] aporta automatismos como la asociación automática entre columnas de las tablas y atributos de las clases, finders dinámicos y reglas de validación. Todos estos automatismos que se generan por defecto han permitido que tan solo se haya tenido que pensar en la lógica de negocio de la aplicación.

En el desarrollo de las clases del modelo se ha seguido siempre la misma metodología: primero se han establecido las relaciones entre modelos, seguidamente si hacía falta se realizaban las validaciones pertinentes y finalmente se programaba la lógica de negocio de la aplicación que recaía en cada clase.

Como ejemplo se muestra la clase Spirometry.

Según el modelo de datos la Espirometría (Spirometry) tiene un detalle (Spirobriefing), un archivo xml (Spirofile), una o varias evaluaciones (Spiroevaluations) y varias maniobras (Spiromaneuvers). También se ha definido que una espirometría pertenece a un tipo de espirometría (spirotype) y pertenece a un profesional, a un paciente y a un centro.

```
class Spirometry < ActiveRecord::Base
  require 'nokogiri' #librerias de tratamiento de XML

  has_one :spirobriefing
  has_one :spirofile

  has_many :spiromaneuvers
  has_many :spiroevaluations

  belongs_to :spirotype
  belongs_to :professional
  belongs_to :patient
  belongs_to :center
```

Ilustración 22 - código - Spirometry relaciones modelo

También se realizan una serie de validaciones básicas, que aseguran que los datos que se guardan en el modelo de datos son correctos.

```
#validaciones basicas
validates_presence_of :professional_id, :patient_id, :center_id

#callbacks

after_create :createbriefing #creamos los detalles de espirometria
```

Ilustración 23 - código - Spirometry validaciones básicas

Finalmente se ha programado la lógica de negocio de la entidad Spirometry. Entre otras podemos encontrar los métodos que devuelven las características socio demográficas del centro, paciente o profesional, finalmente también se encuentran los métodos para obtener los detalles de la espirometría o de sus maniobras.

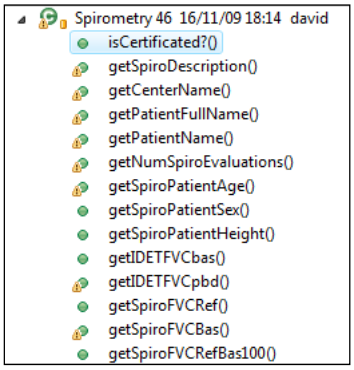



Ilustración 24 - lista de métodos clase Spirometry

```
#Metodos publicos
  def isCertificated?
    return self.certificated
  end
  def getSpiroDescription
    return (self.evaluated.to_s + " " + self.getPatientName + " " + self.getCenterName
    out = "No Hay descripcion de la espirometria"
  end

  def getCenterName

    return self.center.centername unless (self.center.blank? || self.center.centername.
    out = "-"
  end

  def getPatientFullName
    return (self.patient.getPatientName + " " +self.patient.getSurname1) unless (self.p
    out = "-"
  end
  def getPatientName
    return (self.patient.getPatientName ) unless (self.patient.blank?)
    out = "-"
  end

  def getNumSpiroEvaluations
    spsize = 0
    self.spiroevaluations.size unless self.spiroevaluations.size.nil?
  end

  #obtencion de detalles Espirometria-Paciente
  def getSpiroPatientAge

      out = "no hay datos"
      return out if self.spirobriefing.blank?
      return self.spirobriefing.spirob_age unless self.spirobriefing.spirob_age.blank?

  end
```

Ilustración 25 - métodos clase Spirometry

### 3.1.3 Action Controller-Controladores

El enlace entre las Vistas y el Modelo y de las peticiones enviadas desde el navegador se han implementado en las clases controlador. Han sido creados 9 controladores que gestionan todas las acciones de la aplicación. Estos controladores son los de centros, pacientes, profesionales, espirometrías, archivos, evaluaciones, los controladores de usuario y sesión que son creados por la librería de Restful Authentication para la gestión de los usuarios y finalmente se ha creado el application controller que es el controlador común de toda la aplicación del cual todas las clases lo heredan.

El proceso de desarrollo de los controladores se ha dividido en tres fases: En una primera se han programado los filtros, seguidamente se han desarrollado las acciones y finalmente se ha programado el enrutamiento de las peticiones web a los métodos del controlador.

**Definición de filtros**: Todos los controladores comparten el mismo método de filtro que se encarga de verificar que el usuario se ha autentificado en la aplicación y en consecuencia existe una sesión establecida.

**Desarrollo de Controladores**: Cada controlador gestiona las acciones correspondientes a su modelo. La implementación del método correspondiente a la acción del controlador tan solo redirige hacia otras acciones o simplemente crea las instancias necesarias para pasar a las Vistas de la aplicación. Toda la comunicación entre controlador y vista se hace a través de variables de instancia que son compartidas por Vistas y Controladores.

Como ejemplo se muestra la clase del controlador P*rofessionalsController*. Se han creado los métodos de gestión de las acciones *new* y *create* para crear nuevos profesionales, *show* para visualizar los detalles de un profesional y *graphProfSpiros_code* con el que se gestiona la generación de gráficas del profesional.

```ruby
class ProfessionalsController < ApplicationController
  before_filter :login_required, :only => [ :show ]

  def new

    @prof = Professional.new
    @prof.build_user unless @prof.user #creamos el objeto hijo User
    @prof.build_person unless @prof.person #creamos el objeto hijo Person


    @rollist = Proftype.getproftypes
    @centertypelist = Centertype.getcentertypelist
    @centerlist = Center.getcenterlist


  end

  def create

    @prof = Professional.new(params[:professional])
    logger.debug "C-#{self.class} CREATE: #{@prof.inspect}"

    if @prof.save
     render :action => 'new'
    else
      flash[:error]  = "Hay algun problema en el formulario"
      render :action => 'new'
    end

  end

  def show
    #El profesional es el que ha hecho login
    if !current_user.professional.coordinator?
      @prof = @current_user.professional
    else
      @prof = Professional.find(params[:id])
    end
```

Ilustración 26 - código - Controlador – Profesionales

**Routing de la aplicación**: El lugar donde se define el enlace entre peticiones del navegador y las *actions* (métodos de los controladores) es en el archivo Routes.rb.

```ruby
ActionController::Routing::Routes.draw do |map|

  ##### Routes especificas del plugin Acts as authenticate ####
  map.logout '/logout', :controller => 'sessions', :action => 'destroy'
  map.login '/login', :controller => 'sessions', :action => 'new'
  map.register '/register', :controller => 'users', :action => 'create'
  map.signup '/signup', :controller => 'users', :action => 'new'
  map.resources :users
  map.resource :session

  ##### Routes aplicacion ####
  map.resources :spirofiles
  map.resources :spirometries, :has_one => [ :spirobriefing, :spirofile ], :has_many => [ :spiromaneuvers,
  map.resources :professionals, :has_one => [ :person, :user], :has_many => [ :spirometries, :spiroevaluat
  map.resources :proftype, :has_many => :professionals
  map.resources :centers, :has_many => [ :professionals, :patients, :spirometries, :centercontacts, :cente
  map.resources :patients, :has_one => [ :person, :patientdetail], :has_many => [ :spirometries]
```

Ilustración 27 - código - routing aplicación

Se ha definido el mapeo a los recursos de la aplicación mediante el método *map.resources <nombre recurso>*. También se ha definido la herencia entre los distintos recursos con lo que se han creado automáticamente las múltiples rutas para acceder a un mismo recurso.

Por ejemplo se puede acceder al método show del controlador *SpirometriesController* a través de una petición GET a **/spirometries** o a través de un paciente realizando una petición GET a **/patients/:patient_id/spirometries.**

La herramienta Rake de Ruby on Rails permite listar fácilmente todas las rutas disponibles en la aplicación. A continuación se muestra un extracto de las rutas de la aplicación, para ver la lista completa ver anexos (Anexo AC).

```
          logout        /logout                                    {:action=>"destroy", :controller=>"sessions"}
           login        /login                                     {:action=>"new", :controller=>"sessions"}
        register        /register                                  {:action=>"create", :controller=>"users"}
          signup        /signup                                    {:action=>"new", :controller=>"users"}
           users GET    /users(.:format)                           {:action=>"index", :controller=>"users"}
                 POST   /users(.:format)                           {:action=>"create", :controller=>"users"}
        new_user GET    /users/new(.:format)                       {:action=>"new", :controller=>"users"}
       edit_user GET    /users/:id/edit(.:format)                  {:action=>"edit", :controller=>"users"}
            user GET    /users/:id(.:format)                       {:action=>"show", :controller=>"users"}
                 PUT    /users/:id(.:format)                       {:action=>"update", :controller=>"users"}
                 DELETE /users/:id(.:format)                       {:action=>"destroy", :controller=>"users"}
    new_session GET    /session/new(.:format)                     {:action=>"new", :controller=>"sessions"}
   edit_session GET    /session/edit(.:format)                    {:action=>"edit", :controller=>"sessions"}
        session GET    /session(.:format)                         {:action=>"show", :controller=>"sessions"}
                PUT    /session(.:format)                         {:action=>"update", :controller=>"sessions"}
                DELETE /session(.:format)                         {:action=>"destroy", :controller=>"sessions"}
                POST   /session(.:format)                         {:action=>"create", :controller=>"sessions"}
     spirofiles GET    /spirofiles(.:format)                      {:action=>"index", :controller=>"spirofiles"}
                POST   /spirofiles(.:format)                      {:action=>"create", :controller=>"spirofiles"}
  new_spirofile GET    /spirofiles/new(.:format)                  {:action=>"new", :controller=>"spirofiles"}
 edit_spirofile GET    /spirofiles/:id/edit(.:format)             {:action=>"edit", :controller=>"spirofiles"}
      spirofile GET    /spirofiles/:id(.:format)                  {:action=>"show", :controller=>"spirofiles"}
                PUT    /spirofiles/:id(.:format)                  {:action=>"update", :controller=>"spirofiles"}
                DELETE /spirofiles/:id(.:format)                  {:action=>"destroy", :controller=>"spirofiles"}
   spirometries GET    /spirometries(.:format)                    {:action=>"index", :controller=>"spirometries"}
                POST   /spirometries(.:format)                    {:action=>"create", :controller=>"spirometries"}
 new_spirometry GET    /spirometries/new(.:format)                {:action=>"new", :controller=>"spirometries"}
edit_spirometry GET    /spirometries/:id/edit(.:format)           {:action=>"edit", :controller=>"spirometries"}
     spirometry GET    /spirometries/:id(.:format)                {:action=>"show", :controller=>"spirometries"}
                PUT    /spirometries/:id(.:format)                {:action=>"update", :controller=>"spirometries"}
                DELETE /spirometries/:id(.:format)                {:action=>"destroy", :controller=>"spirometries"
```

**Ilustración 28 - Routing App - resumen.**

### 3.1.4 ActionView (AV)-Vistas

Al ser una aplicación Web, la interacción con el usuario se realiza mediante un navegador Web que interpreta las páginas HTML enviadas desde el servidor.

En el presente proyecto se ha utilizado el modulo del Framework Ruby on Rails llamado ActionView. Gracias a ActionView se han podido generar una serie de plantillas HTML a las que se le ha inyectado código Ruby, generando así HTML dinámicos.

Utilizando la terminología del patrón MVC, las vistas generan archivos HTML.ERB[xiii] (plantillas HTML con código Ruby incrustado) las cuales son la representación de la información que se ha enviado como respuesta a la petición de los usuarios.

En el proyecto todas las vistas se agrupan dentro de la carpeta 'espiroq/app/views/' donde cada controlador tiene su propia carpeta y en ella se guardan todas las plantillas RHTML, quedando la siguiente estructura 'espiroq/app/views/controlador/acción.html.erb'.

#### *3.1.4.1 HTML y ERB*

Como ya se ha mencionado anteriormente el diseño de la aplicación se ha realizado mediante el uso de plantillas HTML con código Ruby incrustado en ellas.

Ruby on Rails ha permitido utilizar una plantilla especial llamada Layout. El layout es una plantilla común a toda la aplicación donde según la petición recibida se incrusta la plantilla de la vista solicitada. Se podría decir que el layout ha sido el envoltorio común de toda la aplicación.

[xiii] Embedded Ruby (Ruby incrustado)

En el HTML del layout se han definido los siguientes elementos comunes de la aplicación: cabecera, barra de navegación, y pie de página. Tal como se puede comprobar en las siguientes imágenes, la estructura de la aplicación es siempre la misma.


Ilustración 29 - imagen - layout 1

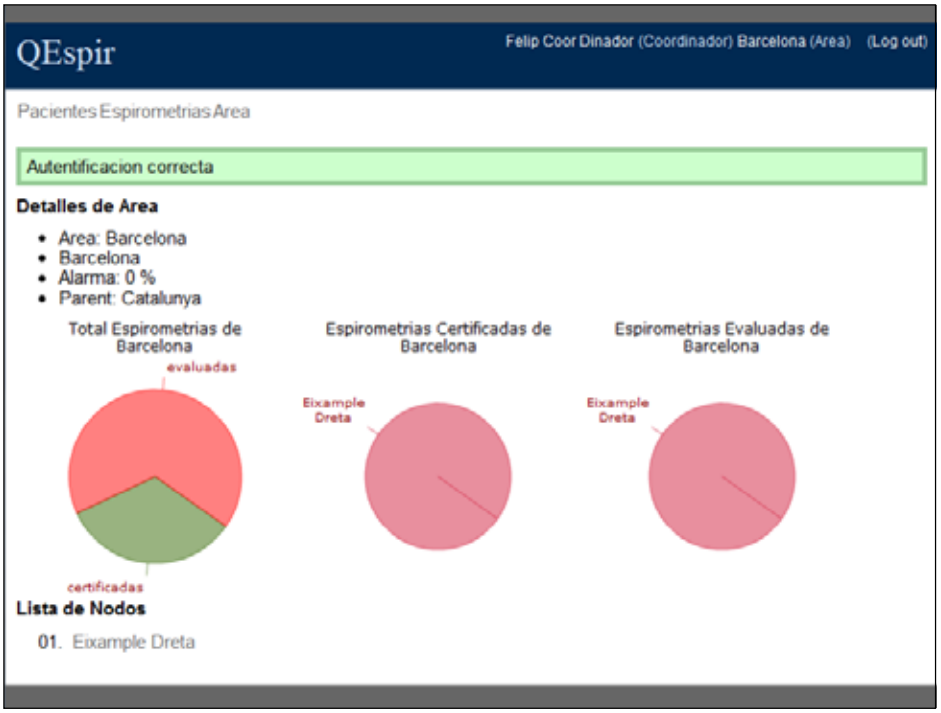



Ilustración 30- imagen - layout 2

### 3.1.4.2 *Maquetación por Capas*

Dentro de la maquetación HTML[xiv] existen dos técnicas de ordenación de elementos dentro de la pantalla. La maquetación por tablas y la maquetación por capas. La primera es una técnica

[xiv] HyperText Markup Language (*Lenguaje de Marcas de Hipertexto*)

obsoleta donde se crean tablas anidadas unas dentro de otras para colocar los elementos de la Web, a partir de la aparición de las CSS[xv] ésta técnica dejo de usarse para empezar a utilizar la maquetación por capas donde los elementos de la Web se separan en capas (DIVS[xvi]) que a través de las hojas de estilo se indica al navegador como han de posicionarse.

En la aplicación la maquetación de las plantillas HTML se ha realizado usando la división por capas y posicionado con CSS.

```
<body>
    <div id="container">
        <div id="header" class="boxMain">

        <!-- Si el usuario no se ha logueado no se muestra el menu-->
        <% unless current_user.blank? %>
            <!-- Segun el rol del prof. se muestra un menu o otro-->
            <% if current_user.coordinator?%>
                <div id="navigation_menu">
            <% else %>
                <div id="navigation_menu">
            <% end %>


        <% end %><br />

        <div id="main" class="boxMain">
        <div id="footer" class="boxMain">
        <!-- Información del pie de pagina-->
        </div>
    </div>
</body>
```

Ilustración 31 - código - separación por capas

### 3.1.4.3 Hojas de estilo CSS

La maquetación de la aplicación se ha realizado mediante el uso de hojas de estilo, también llamadas CSS (*Cascading Style Sheets)* definido por el *World Wide Web Consortium (W3C)* [38].Gracias al diseño mediante CSS se ha realizado un estilismo coherente en toda la aplicación web que es fácilmente modificable.

Dentro de la maquetación de las plantillas HTML con CSS se ha seguido la técnica llamada del 'Modelo de Cajas'. Ésta técnica define todos los elementos de la página web como si fueran cajas rectangulares. Y mediante las hojas de estilo se indica a los navegadores como se han de posicionar dentro la pantalla.

[xv] Cascading Style Sheets (CSS) – *Hojas de estilo en cascada*

[xvi] Etiqueta del lenguaje de programación HTML

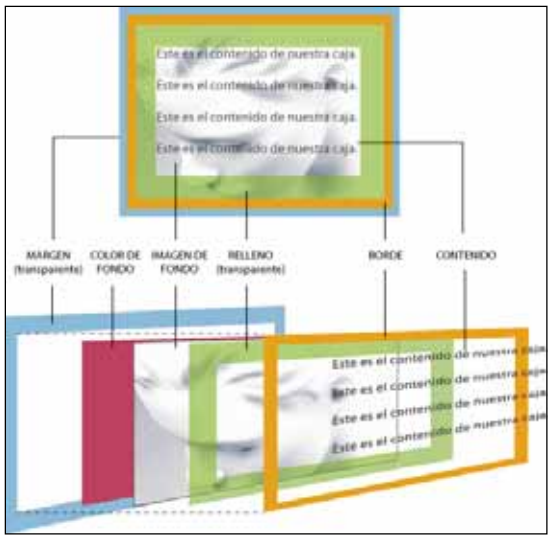



Ilustración 32 - CSS - modelo de cajas

Para aplicar los estilos CSS a toda la aplicación primero se han definido los estilos de los componentes generales (cabeceras, títulos, enlaces, parágrafos, etc.) y después se han creado una serie de clases de estilos que se han aplicado en los diferentes componentes de la aplicación.

```
/****** CLASSES GENERICAS ******/
/*  margin:top right bottom left */

/****** body, a, p, h1, etc. *****/
h1, h2, h3 {
    font-family: tahoma;
    color: #333;
    margin-top: 5px;
    margin-bottom: 5px;
}

em {
    font-size: 1em;
    color: #2F4F4F;
}

/**** Caja estandar ****/
.caja{
    margin: 10px 10px 10px 10px; /* top right bottom left */

}
.boxMain{
    /* margin: 10px 10px 10px 10px; */
    clear: both;
}
.floatLeft{
    float:left;
}

/**** Enlaces ****/
a{text-decoration:none;} /* all links */
a:link {color:#696969;}     /* unvisited link */
a:visited {color:#696969;} /* visited link */
a:hover {color:#0000CD;}    /* mouse over link */
a:active {color:#C0C0C0;}  /* selected link */
```

Ilustración 33 - CSS - Clases

### 3.1.4.4 Partials

Con el objetivo de rehusar al máximo el código escrito, se ha trabajado con los llamados 'partials' que no son más que trozos de código html.erb que pueden ser insertados en cualquier vista.

En el proyecto se pueden encontrar partials en las vistas de users, center, menus, professionals, sessions, spiroevaluations, spiromaneuvers, spirometries y users por lo que generar una nueva vista formada por pequeños componentes de otras vistas es relativamente sencillo y rápido.

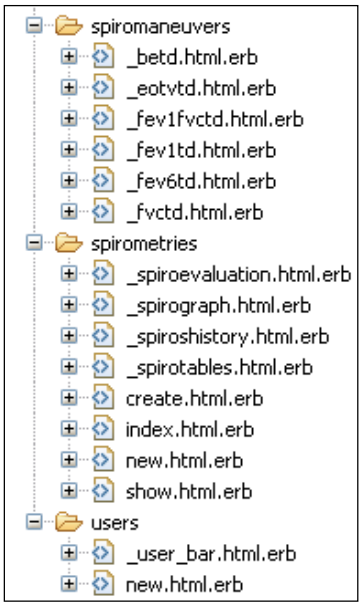



Ilustración 34 - lista partials

Por ejemplo la presentación de la vista correspondiente a la acción show, se ha generado a partir de partials. Se inserta el código HTML de las tablas con los valores de las variables de la espirometría, de las graficas de las curvas, del listado de evaluaciones y, en caso de que se tengan suficientes permisos, del formulario para evaluar la espirometría en cuestión. Se puede comprobar cómo en la plantilla show queda un código muy limpio y fácil de seguir.

```
<h1>Detalles espirometria</h1>
<ul class="ulist" class="boxMain">
    <li>Profesional: <%= link_to @spiro.professional.getfullname, professional_path(@spiro.professional) %></li>
    <li>Paciente: <%= link_to @spiro.patient.
    <li>Evaluada: <%= link_to @spiro.evaluate
    <li>Fecha Espirometria: <%= @spiro.create
</ul>
<ul class="ulist" class="boxMain">
    Detalles del Paciente
    <li>Edad: <%=@spiro.getSpiroPatientAge %>
    <li>Sexo: <%=@spiro.getSpiroPatientSex %>
    <li>Altura: <%=@spiro.getSpiroPatientHeig
</ul>

<ul class="ulist" id="spiroevaluations_histor
    Validaciones
    <% if @spiro.spiroevaluations.empty? %>
        <em>Esta espirometria todavia no ha s
    <% else %>
        <%= render :partial => 'spiroevaluati
    <% end %>
</ul>
<div id="spiro_tablas" class="boxMain">
    <%= render :partial => "spirotables"
</div>
<div id="espiro_graficas" class="boxMai

    <%= render :partial => "spirograph",
</div>

<% if (current_user.professional.coordinator
    <div id="espiro_evaluaciones" class="boxM
        <%= render :partial => "spiroevaluati
    </div>
<% end %>
```

```
<div id="tabla1">
    <table cellspacing="0" class="tablaespiro">
        <tbody>
            <tr class="principal">
                <th>Resultados</th>
            </tr>
            <%#TODO: Acabar SHOW de la tabla de Espirometrias%
            <tr class="titulos_tabla">
                <th></th>
                <th abbr="Valores Referencia">V.Ref.</th>
                <th abbr="Valor Observado">V.Obs.</th>
                <th abbr="% Valor de Referencia">% V.R.</th>
                <th abbr="Post Bronco Dilatador">PBD</th>
                <th abbr="% Post Bronco Dilatador">% PBD</th>
            </tr>
            <tr>
                <th>FVC (L)</th>
                <td><%=@spiro.getSpiroFVCRef%></td>
                <td><%=@spiro.getSpiroFVCBas%></td>
                <td><%=@spiro.getSpiroFVCRefBas100%></td>
                <td><%=@spiro.getSpiroFVCPbd%></td>
                <td><%=@spiro.getSpiroFVCRefPbd100%></td>
            </tr>
            <tr>
                <th>FEV1 (L)</th>
                <td><%= @spiro.getSpiroFEV1Ref%></td>
                <td><%= @spiro.getSpiroFEV1Bas%></td>
                <td><%= @spiro.getSpiroFEV1RefBas100%></td>
                <td><%= @spiro.getSpiroFEV1Pbd%></td>
                <td><%= @spiro.getSpiroFEV1RefPbd100%></td>
            </tr>
```

Ilustración 35 - código - Vista - uso de partials

### 3.1.5 Sesiones y Autentificación

El sistema de sesiones y autentificación de la aplicación se ha delegado en la librería de Ruby on Rails *Restful Authentication*. [39]**.** La librería proporciona tanto altos mecanismos de seguridad en la gestión de usuarios como un eficiente control de sesiones de usuario.

En la implementación del sistema de autenticación se ha creado la clase del modelo User que gestiona la información básica necesaria para el registro (nombre de usuario y contraseña), realiza algunas validaciones y también gestiona la encriptación de la contraseña.

```
require 'digest/sha1'

class   User < ActiveRecord::Base
  include Authentication
  include Authentication::ByPassword
  include Authentication::ByCookieToken

  belongs_to :professional
  #TODO: activar belongs_to :userlanguage

  validates_presence_of     :login
  validates_length_of       :login,    :within => 3..40
  validates_uniqueness_of   :login
#  validates_format_of       :login,    :with => Authentication.login_regex, :message => Authentication.bad_login_message

  validates_format_of       :name,     :with => Authentication.name_regex,  :message => Authentication.bad_name_message, :allow_nil => true
  validates_length_of       :name,     :maximum => 100

  validates_presence_of     :email
  validates_length_of       :email,    :within => 6..100 #r@a.wk

  attr_accessible :login, :email, :name, :password, :password_confirmation

  # Authenticates a user by their login name and unencrypted password.  Returns the user or nil.
  #
  def self.authenticate(login, password)
    return nil if login.blank? || password.blank?
    u = find_by_login(login.downcase) # need to get the salt
    u && u.authenticated?(password) ? u : nil
  end
```

Ilustración 36 - código - Modelo User

La encriptación de la contraseña se realiza aprovechando la librería de Ruby 'digest/sh1' la cual implementa los métodos necesarios para producir una encriptación con el algoritmo SH1 (*Secure Hash Algorithm*, Algoritmo de *Hash* Seguro).

Las contraseñas no se almacenaran en claro sino que se guardaran los 128 bits (40 carácteres hex) resultado de aplicar la función hash criptográfica SHA-1 a la contraseña password

Cuando el usuario quiera autenticarse proporcionará su login y password donde se aplicará el algoritmo SHA-1 a la contraseña y se comparará con el password encriptado que está en el modelo de datos.

Para que el sistema sea algo más seguro se usa el concepto de "un grano de sal" (salting), el cual antes de aplicar SHA-1 se concatena un string pseudoaleatorio (la "sal") a la contraseña. En la BD por tanto se almacena el nombre de usuario, su "sal" y SHA-1(sal + password)

Debido al ámbito de uso de la aplicación (ámbito hospitalario) solo los usuarios registrados pueden tener acceso a las vistas de la aplicación. Solo hay la excepción de la pantalla de login, que es visible para cualquier usuario. La gestión de acceso, se ha realizado mediante filtros que son métodos que se ejecutan antes de la llamada de los controladores correspondientes.

En el proyecto se pueden encontrar filtros en todos los controladores, que antes de ejecutar cualquier acción llaman al método 'login_required', implementado por la librería restful authenticated, que verifica si existe la sesión de un usuario activo o no.

```
class SpirometriesController < ApplicationController
  before_filter :login_required
```

Ilustración 37 - código - filtro verificar sesión

### 3.1.6 Parseo XML

Para realizar el parseo de los archivos XML se ha utilizado una librería de Ruby llamada Nokogiri [40]. Entre otras características el uso de Nokogiri permite las búsquedas en documentos XML a través del lenguaje XPath [41](XML Path Language) y la validación de estructuras de documentos XML.

Toda importación de datos al modelo de datos, necesita una validación previa de los datos que se van a guardar para no poner en peligro la integridad del modelo. En este proyecto la validación es realizada justo antes de hacer persistente el objeto Spirofile.

Para realizar la validación se compara el documento con la definición de la estructura del XML schemasFinalXml.xsd (ver Anexos A.D) con el documento XML que se quiere importar. Si la validación es exitosa se devuelve un flag 'true' y se continúa con el proceso de mapeo de la información.

```
def schemaNDDok?
  logger.debug "M-Spirofile schemaNDDOK? #{self.inspect}"


  xsd = Nokogiri::XML::Schema(File.read("#{RAILS_ROOT}/public/xmls/schemasFinalXml.xsd"))
  doc = Nokogiri::XML(self.spirobinary)


  unless xsd.valid?(doc)
    #mostramos los errores generados en la consola
    xsd.validate(doc).each do |error|
      puts error.message
      logger.debug error.message
      errors.add_to_base(error.message)
    end
   #Añadimos el mensaje de error al objeto ERROR para mostrar en las vistas
   errors.add_to_base("El archivo XML no esta bien creado")
  end
  logger.debug "M-Spirofile schemaNDDOK?_out"
  logger.debug "M-Spirofile schemaNDDOK? value : #{xsd.valid?(doc)}"
  #retorna true si es valido false si no pasa la validacion
  xsd.valid?(doc)

end
```

Ilustración 38 - código - validación Schema

Tras asegurar que el documento XML está correctamente formado se procede a mapear los datos del XML importado al modelo de datos de la aplicación. Este mapeo se realiza en dos fases.

En una primera fase se mapea la información general de la espirometría.

```
def parseNDDXMLtoBriefing

  doc = Nokogiri::XML(self.spirobinary)



  #usamos expresiones xpath para acceder a los elementos del XML
  #.//SN -> accedo al hijo del nodo RAIZ (./), nodo que se llama SN (/SN)

  self.spirometry.spirobriefing.spirob_date = doc.xpath('.//DateTime').text
  self.spirometry.spirobriefing.spirob_vreffvc = doc.xpath('.//Predicted/FVC').text
  self.spirometry.spirobriefing.spirob_vreffev1 = doc.xpath('.//Predicted/FEV1').text
  self.spirometry.spirobriefing.spirob_vreffevfvc = doc.xpath('.//Predicted/FEV1FVC').text
  self.spirometry.spirobriefing.spirob_vreffev6 = doc.xpath('.//Predicted/FEV6').text
  self.spirometry.spirobriefing.spirob_bestfvc = doc.xpath('//TrialData/Results/FVC')[0].text
  self.spirometry.spirobriefing.spirob_bestfev1 = doc.xpath('//TrialData/Results/FEV1')[0].text
```

Ilustración 39 - código - extracto parseo XML - 1

Y tras esta acción se realiza el parseo de la información de cada maniobra.

```
def parseNDDXMLtoManeuvers

  doc = Nokogiri::XML(self.spirobinary)
  i = 0 #contador para recorrer xml
  nummaneuvers = self.spirometry.spiromaneuvers.size #obtenemos el numero de maniobras


  self.spirometry.spiromaneuvers.each do |man|
    logger.debug "M-#{self.class} parseNDDXMLtoManeuvers. i:#{i} - id: #{man.id.inspect}"


   #obtenemos las curvas y sus tamaños codificadas en BASE64 y DeltaComprimidas
    fvCurve = doc.xpath('//TrialData/FVCurve')[i].text
    fvlenght = doc.xpath('//TrialData/FVLength')[i].text
    vtCurve = doc.xpath('//TrialData/VTCurve')[i].text
    vtlenght = doc.xpath('//TrialData/VTLength')[i].text

    fvFormated = self.transformCurve(fvCurve)
    vtFormated = self.transformCurve(vtCurve)

    #Mapeamos los valores del XML con el Objeto Spiromaneuver
    man.spirom_curve = fvFormated + "#," + vtFormated
    man.spirom_fvc = doc.xpath('//TrialData/Results/FVC')[i].text
    man.spirom_fev1 = doc.xpath('//TrialData/Results/FEV1')[i].text
```

Ilustración 40 - código - extracto parseo XML - 2

### 3.1.7 Interpretación de las gráficas

Por cada espirometría importada por la aplicación se almacenan un conjunto de maniobras en el modelo de datos. Cada una de las maniobras está compuesta por una serie de variables (FVC, FEV1, FEV/FVC, etc.) y por dos curvas que son las curvas a representar gráficamente.

El conjunto total de maniobras se divide en 2 grupos. El primer grupo es el del conjunto de maniobras Basales y el segundo grupo será el de maniobras PostBroncodilatadoras.

Como se ha dicho por cada maniobra de la espirometría se encuentran 2 curvas diferentes VT y FV, que representan el nivel de volumen (l) de aire expirado por tiempo (s) y el flujo (l/s) de aire por volumen (l).expirado

En los dispositivos NDD Easyone la información de las curvas es exportada en el fichero XML comprimida y codificada.

Tras el proceso de parseo del documento XML la información de las curvas sufre un proceso de decodificación y son almacenadas en la base de datos. Posteriormente antes de generar las gráficas, la información de las curvas es descomprimida mediante un algoritmo de delta descompresión.

### 3.1.7.1 Decodificación en base 64

La codificación Base 64 codifica los archivos en formato de texto ASCII, por lo que resulta más difícil que los archivos queden dañados cuando se envían a través de Internet.

El modulo Base64 [42]de la librería estándar de Ruby 1.8.7 aporta los métodos necesarios para codificar y decodificar datos binarios en base 64.

En el proyecto la clase Spirofile es la encargada de realizar la decodificación de los datos importados del XML.

En el proceso se decodifica el String que contiene la curva codificada y se guarda en una String de caracteres (bytes)

```
#Descodificamos el string de la curva y lo guardamos en un string de caracteres (bytes)
tmp = Base64.decode64(curve)
```

Ilustración 41 - código - decodificacion base 64

Como fase final de la decodificación, se extrae cada byte (8bits) de la cadena y se transforma en un Signed Integer (16 bits)

```
#extraemos cada caracter como un signed Integer
curveDecoded = tmp.unpack('c*')
```

Ilustración 42 - código-transformación Byte a Signed Integer

### 3.1.7.2 Delta decompression

Los datos de las curvas están delta comprimidos. Esto significa que en una serie de números enteros, la diferencia entre dos muestras de enteros consecutivos mayores de 16 bits se guardan como tipo Byte ocupando así 8 bits. Si la diferencia es mayor que un Signed Byte (+127/-128) entonces se usan múltiples Bytes para expresar la muestra.

Por ejemplo:

El array: de enteros 0,133,688 se comprime como:

133,127,127, 127,127,47 donde 0, 133 -> 133, 688-133 = 555 = 127 * 4 + 47

El encargado de realizar la descompresión es el método deltaDecompress de la clase Spirofile, el cual recibe como parámetro la curva comprimida y la devuelve descomprimida.

Para realizar esta descomprensión se ha realizado el siguiente algoritmo:

```
def deltaDecompress(curvecompressed)

i = 0
iValue = 0
curveDecompressed = Array.new


while i < curvecompressed.size do


  if ((curvecompressed[i] == -128) || (curvecompressed[i] == 127))
    while ((curvecompressed[i] == -128) || (curvecompressed[i] == 127)) do
      iValue = curvecompressed[i] + iValue

      i += 1
    end
    i -= 1
  end

  iValue = curvecompressed[i] + iValue
  iiValue = iValue
  curveDecompressed[i] = iiValue

  i += 1
end

return curveDecompressed
```

Ilustración 43- código - Deltadecompresión

### *3.1.7.3 Graficación de las curvas*

La representación gráfica de las curvas de las espirometrías se ha realizado con la ayuda de un potente componente desarrollado en Flash llamado Open Flash Chart (OFC) [43].

Open Flash Chart es un proyecto de código abierto que permite insertar gran variedad de gráficas dentro de páginas Web. El componente Flash insertado en el código HTML recibe los datos mediante un Json enviado desde el servidor.

En la representación de las curvas de las espirometrías se ha utilizado el tipo de grafica llamada Scatter Chart. En primer lugar de la espirometría (instancia de Spirometry) se obtienen sus maniobras (instancias de Maneuver), de las que se extraen sus 2 curvas VT[xvii] y FV[xviii] a través del método getCurves. Paralelamente se ha creado un objeto ScatterLine, al que se le pasaran los pares de puntos de las graficas (X, Y) mediante objetos ScatterValue. Finalmente la grafica ScatterLine es asignada al objeto Open Flash Chart que es serializado y enviado mediante Json [44] al cliente.

[xvii] Volumen/Tiempo

[xviii] Flujo/Volumen

```
@spiro = Spirometry.find(params[:id])
#logger.debug "C-#{self.class} graph_code:@spiro: #{@spiro.inspect}"

maneuvers = @spiro.spiromaneuvers

#creamos el objeto OpenFlashChart vacio
chart = OpenFlashChart.new

#Recorremos el array con las maniobras
b = 0
p = 0

maneuvers.each do |man|
    colour = "#DB1750"
    dot_size = 3

    scatter = ScatterLine.new(colour,dot_size) #creamos el objeto Scatter (grafica X,Y) del OpenFlashChart

    curves = man.getCurves  #obtenemos un Hash con los puntos del eje Y de la curva VT de la maniobra

    data = curves[:vt] if params[:graphtype]=="vt"
    data = curves[:fv] if params[:graphtype]=="fv"
    logger.debug "C-#{self.class} graph_code:data: #{data}"

    #Recorro todo el array de hash(x,y) y preparo los datos para pasarlos al Open Flash Chart
    #logger.debug "C-#{self.class} graph_code:DATA: BEFORE COLLECT!#{data.inspect}"
    data.collect! do |point|
      ScatterValue.new(point[:x], point[:y])
    end

  if man.spirom_tipo == "b"
    #preparamos la curva
    scatter.text = "Curva Basal " + b.to_s #titulo de la curva
    scatter.width = 1 #tamaño de la linea
    scatter.colour = "#5E47#{b}5" #color de la linea
    scatter.dot_size = 1 #tamaño punto

    #scatter.set_step_horizontal = 2

    #introducimos los datos de los puntos en el objeto linea ScatterLine
    scatter.set_values(data)#introducimos los valores de la curva VT de la maniobra

    b = b + 1
```

Ilustración 44 - código - preparación gráficas -1

```
    chart.add_element(scatter)
  end
     logger.debug "C-#{self.class} graph_code:CHART:  AFTER Maniobras! #{chart}"

      #Preparo la grafica VT y FV
    title = Title.new("Grafica VT") if params[:graphtype]=="vt"
    title = Title.new("Grafica FV") if params[:graphtype]=="fv"

    y = YAxis.new
    y.set_range(0,10,1)

    x = XAxis.new
    x.set_range(0,10,1)

    x_legend = XLegend.new("Tiempo (sg)") if params[:graphtype]=="vt"
    x_legend = XLegend.new("Volumen (l)") if params[:graphtype]=="fv"

    x_legend.set_style('{font-size: 20px; color: #778877}')

    y_legend = YLegend.new("Volumen (l)")if params[:graphtype]=="vt"
    y_legend = YLegend.new("Flujo (l/s)")if params[:graphtype]=="fv"

    y_legend.set_style('{font-size: 20px; color: #770077}')


    chart.set_title(title)
    chart.set_x_legend(x_legend)
    chart.set_y_legend(y_legend)
    chart.y_axis = y
    chart.x_axis = x
    chart.set_bg_colour('#FFFFFF')

    render :text => chart.to_s, :layout => false



end
```

Ilustración 45 - código - preparación gráficas -2

### 3.1.8 Representación tabular de los resultados de la espirometría

Citando la Sociedad Española de Neumología y Cirugía Torácica (SEPAR) [5] en su manual de procedimientos. La expresión de los resultados de una espirometría debe realizarse de la siguiente manera:

*"EXPRESIÓN DE LOS RESULTADOS*

*Los resultados de la espirometría deben expresarse en forma numérica y gráfica. Para la expresión numérica suelen utilizarse tres columnas: en la primera se anotan los valores de referencia para cada variable, en la segunda, los valores obtenidos en el paciente, y en la tercera, el porcentaje de los valores medidos con relación a los de referencia."*

Cada espirometría importada por la aplicación está formada por una serie de maniobras, como se ha comentado en anteriores puntos. De está maniobra se guardan una serie de parámetros en el modelo de datos, con los que posteriormente se generaran dos tablas para ser mostradas en las vistas correspondientes. La primera tabla muestra el resultado de la espirometría tomando las mejores maniobras realizadas y la segunda tabla mostrará las variables resultantes de todas las maniobras.

#### *3.1.8.1 Tabla1- resultados mejores maniobras basal y postbroncodilatadora*

La primera tabla reproduce la expresión de resultados sugerida por la SEPAR para las maniobras Basales y Postbroncodilatadoras.

**Vref**: Valor de referencia

**Observado**: Valor obtenido en el paciente en la mejor de las maniobras Basales

**% V.R**: Porcentaje de los valores medidos con relación a los de referencia

**PBD**: Valor obtenido en el paciente en la mejor de las maniobras Postbroncodilatadoras

**%PBD**: Porcentaje de los valores medidos con relación a los de referencia

| Resultados | | | | | |
|---|---|---|---|---|---|
| | V.Ref. | V.Obs. | % V.R. | PBD | % PBD |
| FVC (L) | 5.32 | 4.32 | -81.2 | - | - |
| FEV1 (L) | 4.37 | 3.21 | 73.46 | - | - |
| FEV1/FVC % | 0.82 | 74.31 | -90.47 | - | - |
| FEV6 | 5.27 | 4.14 | -127.29 | - | - |

Ilustración 46 - detalle espirometría - tabla 1

La instancia de la espirometría (clase Spirometry) es la encargada de obtener los datos necesarios para llenar la tabla. Los métodos que intervienen en el proceso son los siguientes:

| Valores de referencia | Valor Observado | % Valor de Referencia | Post Bronco Dilatador | % Post Bronco Dilatador |
|---|---|---|---|---|
| getSpiroFVCRef | getSpiroFVCBas | getSpiroFVCRefBas100 | getSpiroFVCPbd | getSpiroFVCRefPbd100 |
| getSpiroFEV1Ref | getSpiroFEV1Bas | getSpiroFEV1RefBas100 | getSpiroFEV1Pbd | getSpiroFEV1RefPbd100 |
| getSpiroFEV1FVCRef | getSpiroFEV1FVCBas | getSpiroFEV1FVCRefBas100 | getSpiroFEV1FVCPbd | getSpiroFEV1FVCRefPbd100 |
| getSpiroFEV6Ref | getSpiroFEV6Bas | getSpiroRefFEV6Bas100 | getSpiroFEV6Pbd | getSpiroFEV6RefPbd100 |

Tabla 4- Valores Tabla 1 - detalles espirometría

Valores de referencia:

**getSpiroFVCRef**: Retorna IVREFFVC que es el valor de referencia importado del XML. Retorna el valor directamente del modelo de datos.

**getSpiroFEV1Ref**: Retorna IVREFFEV1 que es el valor de referencia importado del XML. Retorna el valor directamente del modelo de datos.

**getSpiroFEV1FVCRef**: Retorna IVREFFEVFVC que es el valor de referencia importado del XML. Retorna el valor directamente del modelo de datos.

**getSpiroFEV6Ref**: Retorna IVREFFEV6 que es el valor de referencia importado del XML. Retorna el valor directamente del modelo de datos.

Valores observados:

**getSpiroFVCBas**: Valor de la variable de la mejor maniobra realizada al paciente. El método retorna IDETFVC que es el valor guardado en el modelo de datos.

**getSpiroFEV1Bas**: Valor de la variable de la mejor maniobra realizada al paciente. El método retorna IDETFEV1 que es el valor guardado en el modelo de datos.

**getSpiroFEV1FVCBas**: Valor de la variable de la mejor maniobra realizada al paciente. El método retorna IDETFEVFVC que es el valor guardado en el modelo de datos.

**getSpiroFEV6Bas**: Valor de la variable de la mejor maniobra realizada al paciente. El método retorna IDETFEV6 que es el valor guardado en el modelo de datos.

% Valores de Referencia

**getSpiroFVCRefBas100:** Porcentaje de valor de referencia y observado. El método retorna "getIDETFVCbas /getSpiroFVCRef * 100"

**getSpiroFEV1RefBas100:** Porcentaje de valor de referencia y observado. El método retorna "getSpiroFEV1Bas/getSpiroFEV1Ref * 100"

**getSpiroFEV1FVCRefBas100:** Porcentaje de valor de referencia y observado. El método retorna "getSpiroFEV1RefBas100/getSpiroFVCRefBas100) * 100"

**getSpiroRefFEV6Bas100:** Porcentaje de valor de referencia y observado. El método retorna "getSpiroFEV6Ref / (-1 * self.getSpiroFEV6Bas) * 100"

El método para obtener los valores de la columna PostBroncoDilatador y %PostBroncoDilatador es el mismo que para las maniobras Basales pero con las maniobras Postbroncodilatadoras.

#### *3.1.8.2 Tabla2- resultados maniobras basal y postbroncodilatadora*

Para cada fila de la segunda tabla las primeras tres columnas corresponden con las tres primeras maniobras o medidas basales (FVC) y las tres últimas columnas corresponden con las tres primeras medidas broncodilatadoras (PBD).

Cada una de las columnas está asociada con una curva (primera curva basal, segunda basal, tercera basal, primera broncodilatadora, segunda broncodilatadora, tercera broncodilatadora) generando un total de seis columnas.

Cada fila representa valores de cada una de las maniobras:

- **FVC**: Capacidad vital forzada.(El mayor volumen de aire que puede ser expulsado de los pulmones en una maniobra forzada)
- **FEV1**: Volumen espirado en el primer segundo.(El mayor volumen de aire que puede ser expulsado de los pulmones en el primer segundo de una espiración forzada)
- **FEV1/FVC**: Relación entre el FEV1 y la FVC.

- **FEV6**: Volumen espirado en los 6 primeros segundos.(El mayor volumen de aire que puede ser expulsado de los pulmones en los 6 primeros segundos de una espiración forzada)
- **BE**: Volumen Extrapolado Anterior
- **EOTV**: Volumen al final del test

Si hubiera menos de seis maniobras en total o curvas (TRIALDATA), habrá columnas vacías.

En todos los casos donde exista valor, éste es el obtenido desde el archivo XML multiplicado por -1.

| | Basal | | | PBD | | |
|---|---|---|---|---|---|---|
| | Maniobra 1 | Maniobra 2 | Maniobra 3 | Maniobra 1 | Maniobra 2 | Maniobra 3 |
| FVC (L) | -4.32 | -4.27 | -4.26 | | | |
| FEV1 (L) | -3.21 | -3.01 | -3.22 | | | |
| FEV1/FVC % | | | | | | |
| FEV6 | -4.14 | -4.07 | -4.24 | | | |
| BE | 0.01 | 0.01 | 0.02 | | | |
| EOTV | 0.01 | 0.01 | 0.02 | | | |

Ilustración 47 - detalle espirometría - tabla 2

Siguiendo el principio DRY (Don't Repeat Yourself) que recomienda seguir la filosofía Rails, la generación de la segunda tabla se ha realizado mediante partials (3.1.4.2). Se han creado 2 grupos de partials, al primer grupo se le pasan las maniobras basales y al segundo grupo se le pasan las maniobras postbroncodilatadoras.

```
<tr>
    <th>FVC (L)</th>
    <%= render :partial => 'spiromaneuvers/fvctd', :collection => @spiro.getSpiroManeuversBas %>
    <%= render :partial => 'spiromaneuvers/fvctd', :collection => @spiro.getSpiroManeuversPbd %>
</tr>
<tr>
    <th>FEV1 (L)</th>
    <%= render :partial => 'spiromaneuvers/fev1td', :collection => @spiro.getSpiroManeuversBas %>
    <%= render :partial => 'spiromaneuvers/fev1td', :collection => @spiro.getSpiroManeuversPbd %>
</tr>
<tr>
    <th>FEV1/FVC %</th>
    <%= render :partial => 'spiromaneuvers/fev1fvctd', :collection => @spiro.getSpiroManeuversBas %>
    <%= render :partial => 'spiromaneuvers/fev1fvctd', :collection => @spiro.getSpiroManeuversPbd %>
</tr>
<tr>
    <th>FEV6</th>
    <%= render :partial => 'spiromaneuvers/fev6td', :collection => @spiro.getSpiroManeuversBas %>
    <%= render :partial => 'spiromaneuvers/fev6td', :collection => @spiro.getSpiroManeuversPbd %>
</tr>
<tr>
    <th>BE</th>
    <%= render :partial => 'spiromaneuvers/betd', :collection => @spiro.getSpiroManeuversBas %>
    <%= render :partial => 'spiromaneuvers/betd', :collection => @spiro.getSpiroManeuversPbd %>
</tr>
<tr>
    <th>EOTV</th>
    <%= render :partial => 'spiromaneuvers/eotvtd', :collection => @spiro.getSpiroManeuversBas %>
    <%= render :partial => 'spiromaneuvers/eotvtd', :collection => @spiro.getSpiroManeuversPbd %>
</tr>
```

Ilustración 48 - tabla2 - codigo partials

Dentro de cada partial, simplemente se recoge el valor correspondiente del modelo de datos y se le añaden las etiquetas HTML de columna.

En el caso del valor FVC de las maniobras basales, el partial contiene el siguiente código.

```
<td><%= fvctd.spirom_fvc%></td>
```

Ilustración 49 - tabla2 -código partial - FVC

### 3.1.9 Gráficas de Gestión

La representación gráfica de las estadísticas de gestión de profesionales y centros se ha realizado utilizando Open Flash Chart el mismo componente que en la representación gráfica de las curvas de las espirometrías (3.1.7.3).

En el proyecto existen dos clases de estadísticas de gestión diferentes, que son gestionadas por diferentes clases del modelo. Estas estadísticas son:

#### *3.1.9.1 Estadísticas de calidad de gestión de profesionales*

La responsabilidad de generar los datos de calidad recae sobre la clase Professional.rb la cual obtiene los datos a través de los métodos: getNumSpiros, getNumSpirosEvaluated, getNumSpirosCertificated y getNumSpirosNoEvaluated.

```ruby
def getNumSpirosEvaluated
  Spirometry.count(:conditions => "professional_id = '#{self.id}' AND evaluated = '1'")
end
def getNumSpirosCertificated
  Spirometry.count(:conditions => "professional_id = '#{self.id}' AND certificated = '1'")
end
def getNumSpirosNoEvaluated
  Spirometry.count(:conditions => "professional_id = '#{self.id}' AND evaluated = '0' AND certificated = '0' ")
end
```

Ilustración 50 - metodos obtención espiros profesional

### *3.1.9.2 Estadísticas de calidad de gestión de de la zonas sanitarias.*

En este caso la responsabilidad de generar los datos de calidad recae sobre la clase Center.rb la cual obtiene los datos a través de los métodos: getAllNumSpirosEvaluated, getAllNumSpirosCertificated y getAllNumSpirosNoEvaluated. En este caso en los algoritmos de obtención de número total de espirometrías se ha evaluado que tipo de zona se trataba, es decir que, si no era un centro (el último nivel sanitario), se ha realizado una consulta a todos los subcentros hasta obtener el número total de espirometrías.

```ruby
#Retorna el numero de espirometrias NO certificadas pero SI evaluadas del centro instanciado
def getNumSpirosEvaluated
  Spirometry.count(:conditions => "center_id = '#{self.id}' AND evaluated = '1'")
end

#Retorna el numero de espirometrias certificadas del centro instanciado
def getNumSpirosCertificated
  Spirometry.count(:conditions => "center_id = '#{self.id}' AND certificated = '1'")
end

#Retorna el numero de espirometrias NO certificadas y NO evaluadas del centro instanciado
def getNumSpirosNoEvaluated
  Spirometry.count(:conditions => "center_id = '#{self.id}' AND evaluated = '0' AND certificated = '0' ")
end
```

Ilustración 51 - métodos obtención espirometrías centro

```
def getAllNumSpirosEvaluated
  numspiros = 0
  numspiros = self.getNumSpirosEvaluated
  if self.morechildren?
    self.getchildren.each do |c|
      numspiros += c.getAllNumSpirosEvaluated
    end
  end
  return numspiros
end
def getAllNumSpirosCertificated
  numspiros = 0
  numspiros = self.getNumSpirosCertificated
  if self.morechildren?
    self.getchildren.each do |c|
      numspiros += c.getAllNumSpirosCertificated
    end
  end
  return numspiros
end
def getAllNumSpirosNoEvaluated
  numspiros = 0
  numspiros = self.getNumSpirosNoEvaluated
  if self.morechildren?
    self.getchildren.each do |c|
      numspiros += c.getAllNumSpirosNoEvaluated
    end
  end
  return numspiros
end
```

Ilustración 52 - métodos obtención espirometrias subcentros

## 3.2 Migración de datos

Con el objetivo de dar un valor añadido al proyecto, se ha realizado una migración de los datos recogidos en el estudio Spir@p a la base de datos de la nueva aplicación. Gracias a este traspaso ha sido posible realizar una simulación real de carga de la aplicación con 4000 registros de espirometrías.

La migración ha sido realizada utilizando la herramienta de Sun microsistems MySQL Migration Toolkit [45], con la que se realizado un volcado de datos entre la base de datos Oracle de la aplicación E-Spiro y la aplicación desarrollada en este proyecto EspiroQ. El mapeo entre tablas se puede consultar en los anexos (ver anexo A.E) .

# 4 CASO REAL

En este apartado se simula un caso real y se muestra, pantalla a pantalla, el flujo seguido desde la realización de la espirometría hasta la certificación final de la espirometría.

El Centro Externo (Centro de Atención Primaria) dependiente de *Servei Catalá de la Salud (Generalitat de Catalunya)* realiza una espirometría con el espirómetro NDD Easyone.

Ilustración 53 - espirómetro NDD

Una vez finalizada la exportación del archivo XML de la espirometría se conecta a través de IP a la aplicación.


Ilustración 54 - EspiroQ - login técnico

El técnico importa el archivo XML generado por el programa del espirómetro.

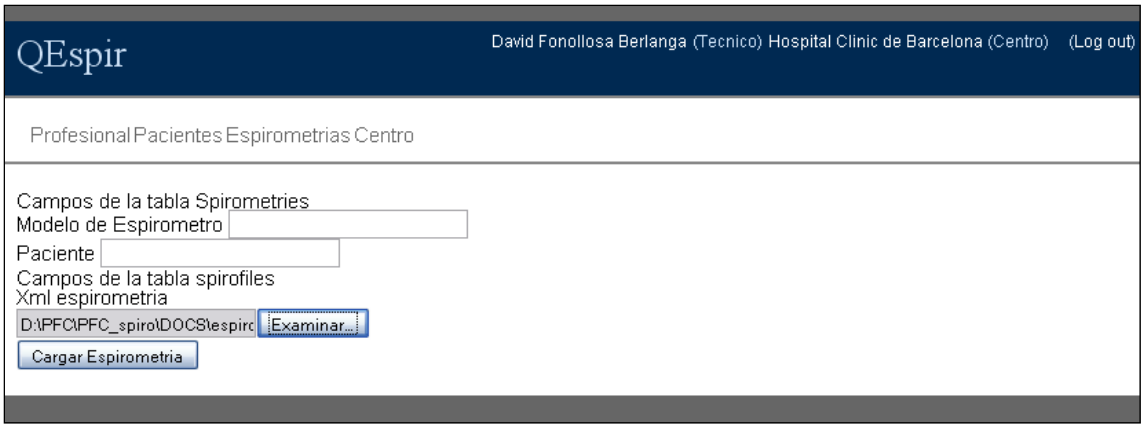



Ilustración 55 - EspiroQ - importación de datos de espirometría a través del archivo XML

Una vez importado el XML y se visualiza tanto los gráficos como los valores numéricos de la espirometría realizada.

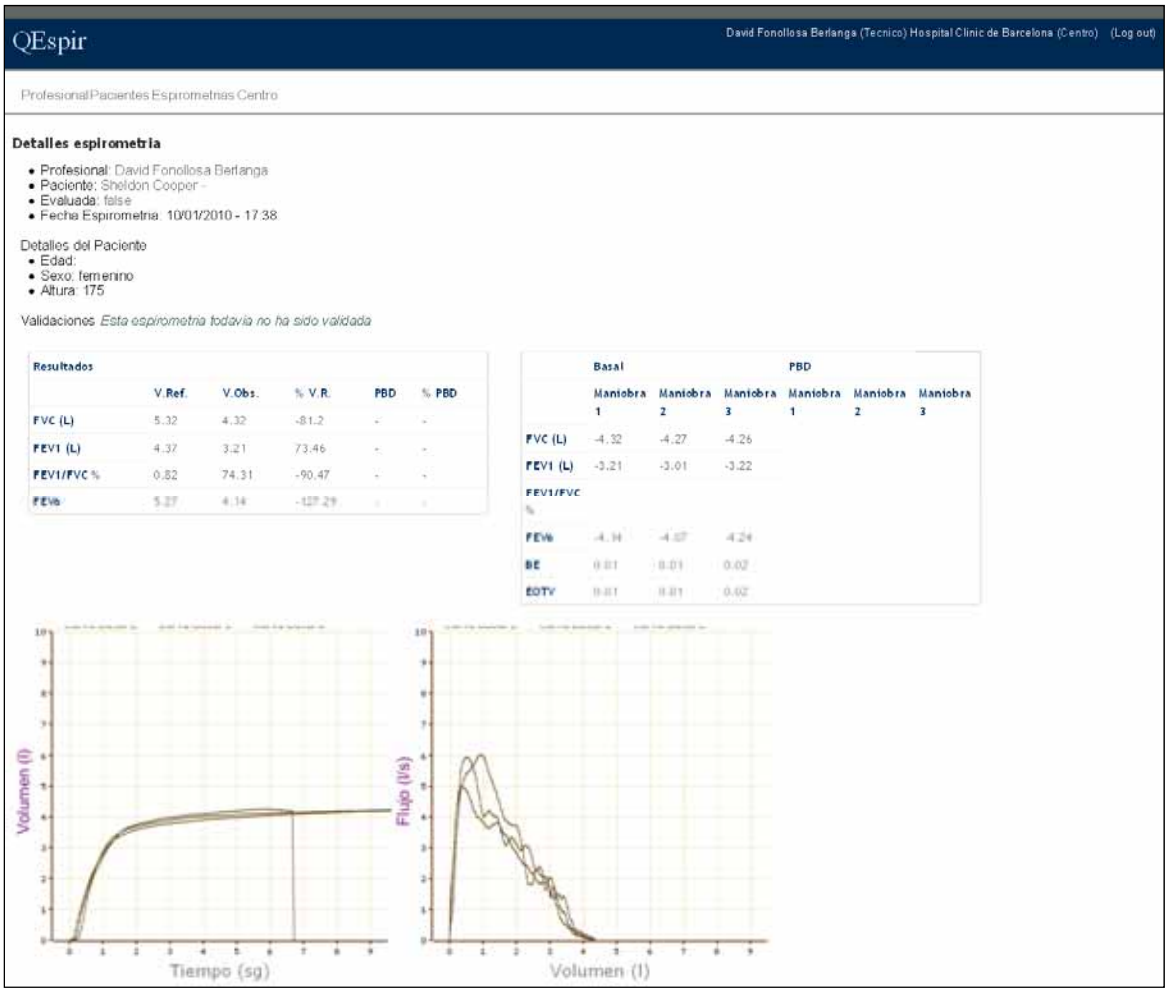



Ilustración 56 - EspiroQ - Valoración Visual de la espirometría

El centro supervisor (Laboratorio de Función Pulmonar de Hospital Clínic) accede a la aplicación.


Ilustración 57 - EspiroQ - login coordinador

El coordinador visualiza las mismas pantallas que el centro externo cualificando la espirometría según su grado de calidad.

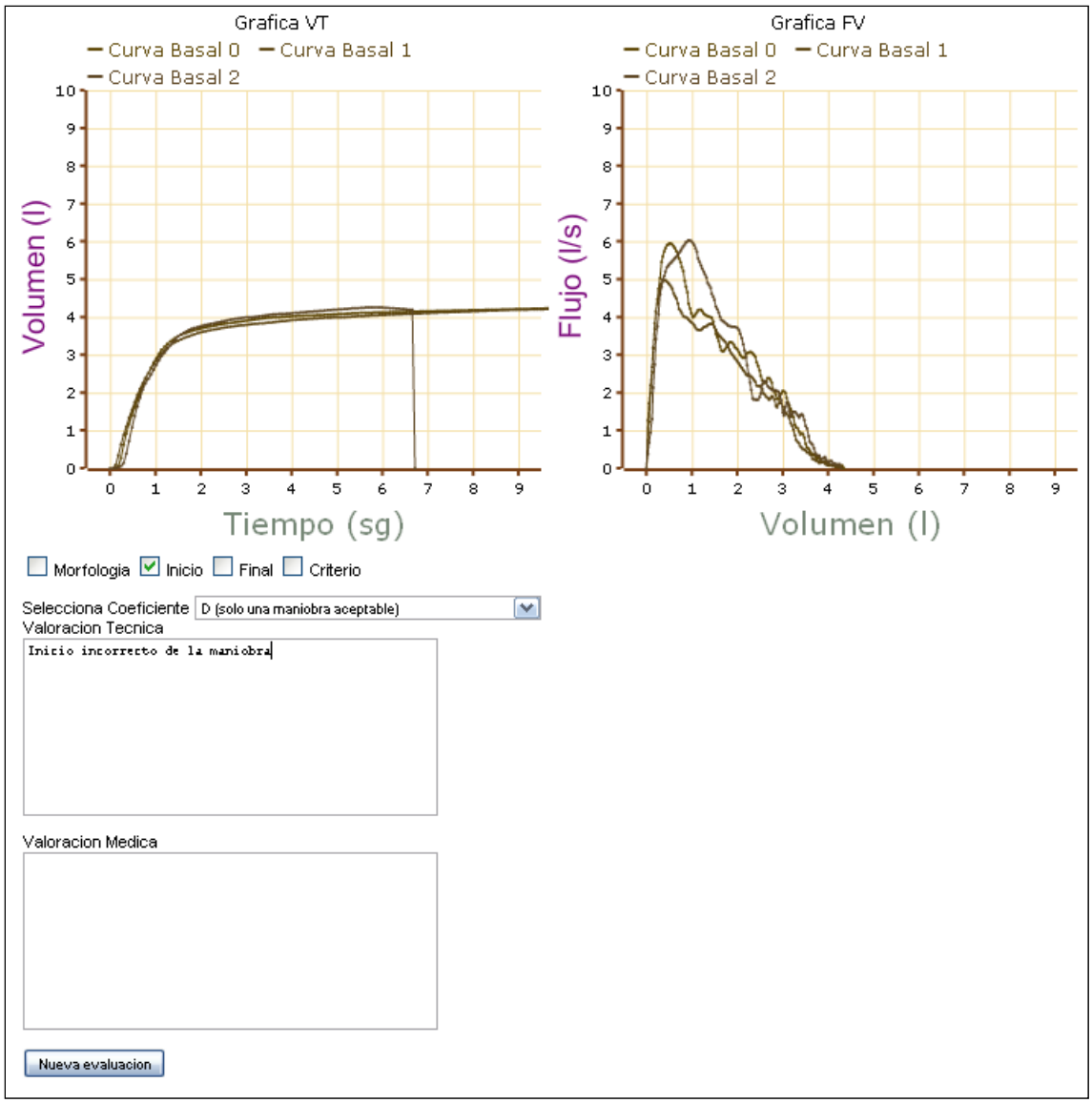



**Ilustración 58 - EspiroQ - Evaluación de calidad**

Así mismo se informa con unas calificaciones estándares sobre los diversos aspectos que influyen en la calidad de la espirometría, en caso de no cumplir los criterios predeterminados se emite un informe con las medidas correctoras. Finalmente si cumple los criterios establecidos se emite un certificado de calidad digital.

Si el Centro de Atención Primaria lo solicita se efectúa un informe médico de la prueba funcional a la que también se le adjunta un certificado digital.

Toda esta información es remitida vía IP al centro externo.

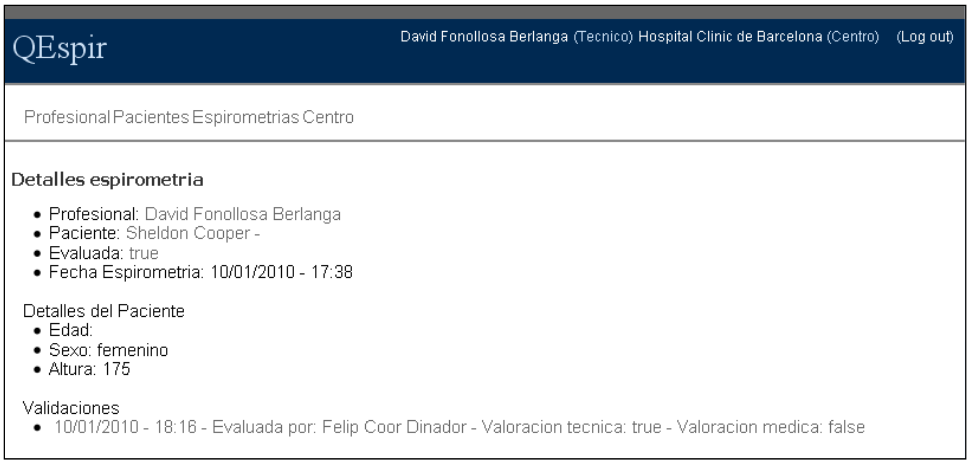



Ilustración 59 – EspiroQ - Detalles generales de la espirometría

Simultáneamente se realiza un análisis de los resultados, desde el punto de vista de calidad, presentado un informe por profesional, por centro o por áreas de gestión.

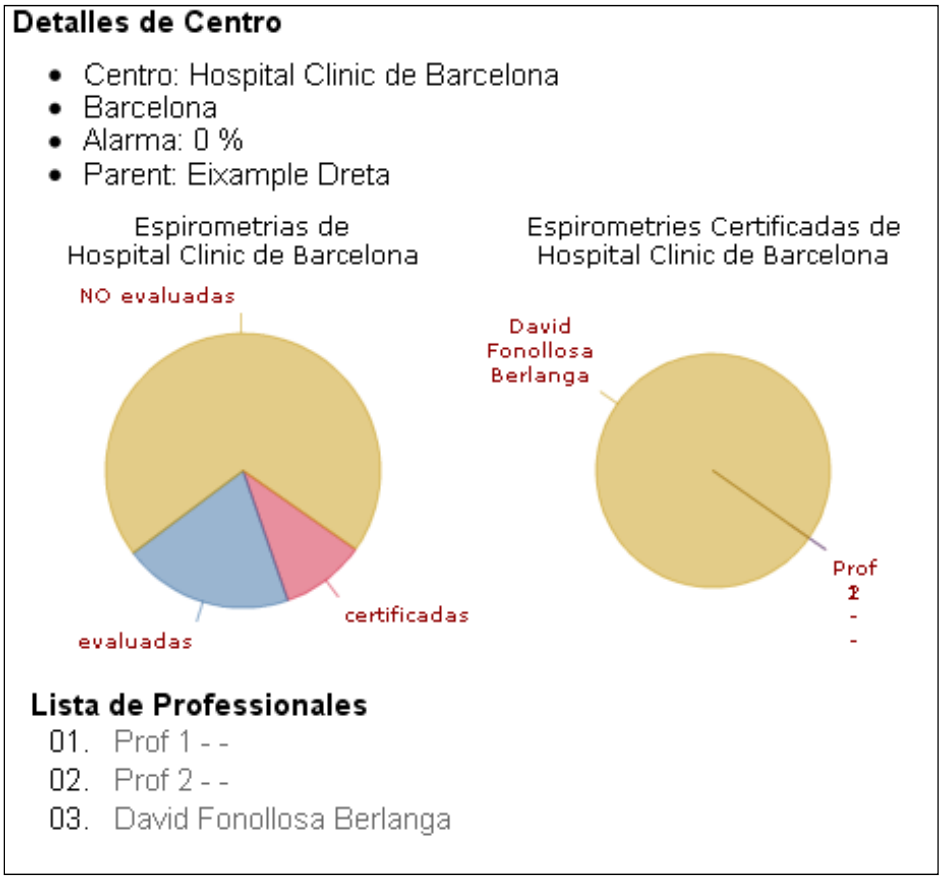



Ilustración 60 - EspiroQ - Gráficas de gestión del Centro

En el análisis también se realiza una consulta de las estadísticas operativas: número total de espirometrías realizadas por el profesional, número de espirometrías correctas y número de espirometrías erróneas.

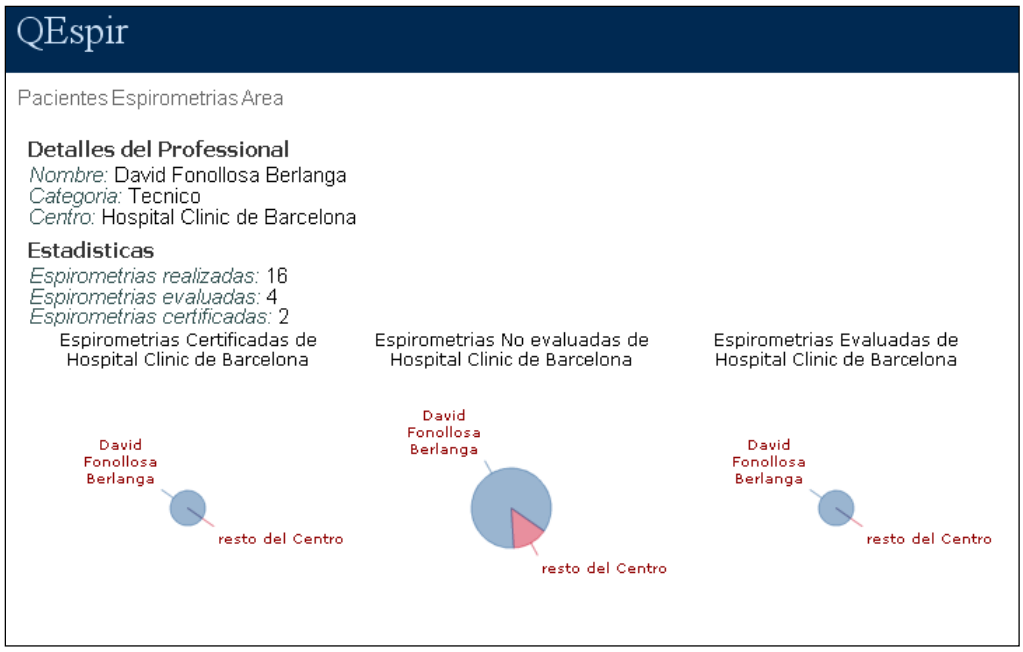



Ilustración 61 - EspiroQ - Gráficas de gestión del profesional

Finalmente en el caso de espirometrías no válidas, en el informe se ha de incluir la razón de la incorrección detectada en la maniobra espirométrica.

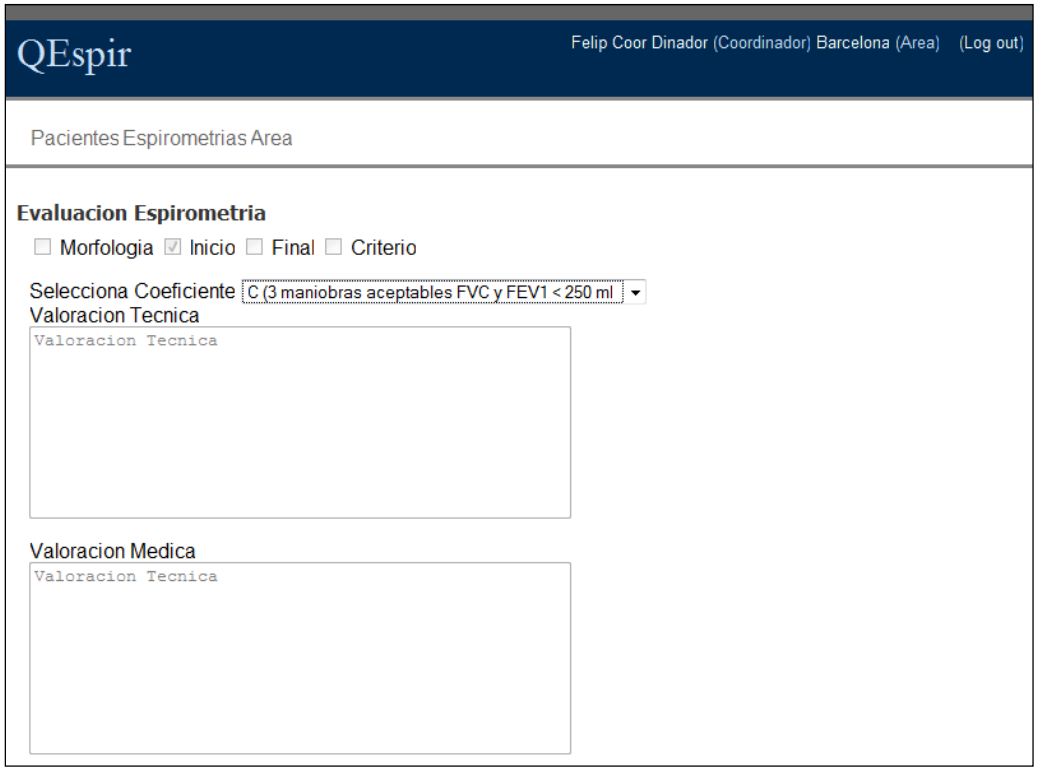



Ilustración 62 - EspiroQ - Consulta de detalles de evaluación realizada

.

# 5 CONCLUSIONES

## 5.1 Conclusiones del proyecto

Se ha diseñado y desarrollado EspiroQ. Una aplicación informática que permitie la importación, visualización y posterior validación de calidad de espirometrias realizadas fuera de los laboratorios de función pulmonar. Permitiendo así una mejora, fuera de los centros especializados, en el proceso de detección de enfermedades pulmonares. Esta optimización del proceso de diagnostico precoz de enfermedades pulmonares supondrá una reducción del flujo de pacientes a ser tratados en los centros hospitalarios, hecho que conllevará mejoras en gestión del tiempo de los profesionales viendose reflejado en un mejor servicio hacia los pacientes y un ahorro económico para los centros hospitalarios.

Hay que remarcar que no existe ninguna aplicación diseñada para valorar espirometrias remotamente, y realizar posteriormente el control de calidad del trabajo de los centros. Por lo que gracias a EspiroQ será posible efectuar estudios en distintos niveles sanitarios, que permitan detectar potenciales puntos de conflicto en el proceso actual de asistencia sanitaria pulmonar fuera de los entornos especializados.

Es destacable comentar que EspiroQ encaja perfectamente dentro de las estrategias de asistencia remota que Las autoridades sanitarias tanto a nivel autonómico como nacional [46] estan realizando para fomentar el uso extensivo de la espirometría forzada de calidad fuera de los laboratorios de función pulmonar. Por lo que se perfila como una herramienta de telemedicina con gran potencial.

EspiroQ va a ser utilizada en una primera fase en un proyecto de prueba de la herramienta a nivel de Atención Primaria. Si se producen resultados satisfactorios esta herramienta podrá formar parte conjunto de medidas destinadas a fomentar el uso extensivo de la espirometría forzada de calidad fuera de los laboratorios de función pulmonar.

Concluir que considero que se ha conseguido el objetivo inicial, desarrollando una herramienta válida para efectuar el control de calidad de las espirometrías realizadas remotamente fuera del entorno hospitalario.

## 5.2 Conclusiones Personales

Este proyecto me ha permitido consolidar y ampliar el conocimiento recibido en la carrera específicamente la relacionada con la rama de telemática. He aplicado sobre todo los conocimientos adquiridos en las asignaturas de Cálculo, Programación, Telemática, Uso de Sistemas Operativos.

He de destacar que a nivel teórico he aprendido una buena base de programación orientada a objetos a través del lenguaje de programación Ruby, que me ha obligado a abstraerme y pensar solo en objetos. El entorno de desarrollo Web Ruby on Rails ha permitido que haya aprendido prácticamente sin querer conceptos como los principios de diseño de bajo acoplamiento y alta cohesión o patrones de diseño como el Modelo – Vista – Controlador.

Ruby on Rails también me ha obligado a seguir buenos principios de diseño en el desarrollo de aplicaciones Web: uso de herramientas de control de versiones, debugar, realizar tests, comentar mucho, escribir código legible, rehusar (DRY), etc. Se puede considerar confirmado que aprender a programar con Ruby on Rails es un buen paso previo a la instrucción en Java.

También he de hablar sobre la metodología de trabajo seguida en éste proyecto que me ha servido para ver lo importante que es realizar un buen trabajo previo de diseño de una aplicación informática. Diría que el diseño de la aplicación es la fase más difícil de realizar, ya que la programación puede tener algoritmos más o menos complicados de implementar, pero si el diseño está mal realizado entonces la aplicación no será útil para el usuario final. Dentro del diseño todos los pasos seguidos han sido muy importantes, pero remarcaría sobre todo la recogida de requisitos donde se ha realizado una verdadera tarea de ingeniería para interpretar las necesidades del usuario final, y también destacaría el diseño del modelo de datos, que se ha diseñado lo mas modular y escalable posible.

Gracias a realizar este proyecto dentro del Hospital Clínic de Barcelona, he aprendido mucho sobre otros ámbitos fuera de la ingeniería de telecomunicaciones. He podido realizar un proyecto que une medicina, enfermería y telecomunicaciones lo que me ha aportado una experiencia fundamental para trabajar a partir de ahora en proyectos de telemedicina.

## 5.3 Trabajos futuros

Como punto final se consideran las líneas futuras de trabajo y posibles mejoras.

Como trabajo futuro se ha de desarrollar las pantallas de administración de centros, pacientes y las de las tablas de tipos. Otro de los puntos a desarrollar es la de ampliar la compatibilidad de la aplicación con la posibilidad de importar XML de nuevos modelos de espirómetros.

También queda pendiente la integración total con la plataforma de telemedicina Linkcare. Llevar a cabo esta tarea no ha de ser complicado, ya que el diseño del modelo de datos en su estructura principal utiliza las mismas entidades que el modelo de datos de la plataforma Linkcare.

Como posible mejora se podría pensar en la implementación del modulo de certificación digital, que implementaría el envío automático por correo electrónico de un documento pdf firmado digitalmente con la validación de la espirometría.

Para concluir comentar que esta aplicación es un programa de telemedicina que todavía no existe, por lo que con la experiencia surgida a partir de su uso por parte de los usuarios van a aparecer nuevos requisitos que probablemente obliguen a cambiar totalmente el diseño inicial de la aplicación.

# 6 Bibliografía


[1] **American Thoracic Society.** *Standardization of spirometry 1994 update.* s.l. : Am J Respir Crit, 1995. 152: 1107-1136.

[2] *Standardisation of spirometry.* **M.R. Miller, J. Hankinson, V. Brusasco, F. Burgos, R. Casaburi, A. Coates, R. Crapo, P. Enright, C.P.M. van der Grinten, P. Gustafsson, R. Jensen, D.C. Johnson, N. MacIntyre, R. McKay, D. Navajas, O.F. Pedersen, R. Pellegrino, G. Viegi, J. Wanger.** 26 (319-338), s.l. : Eur Respir Journal, 2005.

[3] **Roca, J, Sanchis, J and Agustí-Vidal, A.** *Spirometric reference values for a mediterranean.* s.l. : Bull Eur Physiopathol Respir, 1986. 22: 217-224.

[4] *La espirometria en atencion primaria en Navarra.* **Hueto J., Cebollero P., Pascal I., Cascante J.A., Eguia V.M., Teruel F., Carpintero M.** 326-31., s.l. : Arch Bronconeumol, 2006, Vol. 42(7).

[5] **Sociedad Española de Neumología y Cirugía Torácica (SEPAR).** Manual SEPAR de Procedimientos.Módulo 3. Procedimientos de evaluación de la función pulmonar. [En línea] 2002. www.separ.es/doc/publicaciones/historico/procedimientos3.pdf. 84-7989-155-6.

[6] **J. de Miguel Díeza, J.L. Izquierdo Alonsob, J. Molina Parísc, J.M. Rodríguez González-Moroa,.** Fiabilidad del diagnóstico de la EPOC en atención primaria y neumología en España. Factores predictivos. [Online] http://www.archbronconeumol.org.

[7] **Technologies, ndd Medical.** ndd Medical Technologies. *ndd Medical Technologies.* [Online] http://www.ndd.ch/.

[8] Sibel. *Sibelmed.* [Online] http://www.sibelmed.es/.

[9] **Bennett, Purittan.** Renaissance II . [Online] http://www.puritanbennett.com/prod/Product.aspx?id=250.

[10] **Foundation, Apache Software.** Struts 2. [Online] http://struts.apache.org/2.x/.

[11] **Red Hat, Inc.** Hibernate. [Online] https://www.hibernate.org/.

[12] Jaeger - SpiroPro. [Online] http://www.jaeger-toennies.com/english/products/lung-function/spirometer/e-spiro.html.

[13] **Union, European.** Nomenclature of territorial units for statistics . *NUTS Statistical Regions of Europe.* [Online] http://ec.europa.eu/eurostat/ramon/nuts/home_regions_en.html.

[14] **Fundació Clínic per la Recerca Biomèdica.** Fundació Clínic per la Recerca Biomèdica. [Online] http://www.fundacioclinic.org/.

[15] **Welton, David N.** The Economics of Programming Languages. [Online] http://www.welton.it/articles/programming_language_economics.html.

[16] **DedaSys LLC.** Programming Language Popularity . [Online] http://www.langpop.com/.

[17] **craigslist, inc.** Craigslist Classifieds. [Online] http://www.craigslist.org/.

[18] **Dice Holdings, Inc.** Dice.com - Job Search for Technology Professionals. [Online] http://www.dice.com/.

[19] **Indeed.** Indeed. [Online] http://www.indeed.es/.

[20] **Google. inc.** Google Code. [Online] http://code.google.com/intl/es-ES/.

[21] **Geeknet, Inc.** Ohloh, the open source network. [Online] http://www.ohloh.net/.

[22] **Yahoo Company.** Delicious. [Online] http://delicious.com/.

[23] **Yahoo!** Yahoo Search. [Online] http://es.search.yahoo.com/.

[24] freshmeat.net. [Online] http://freshmeat.net/.

[25] **Matsumoto, Yukihiro "matz".** Ruby. *Ruby.* [Online] http://www.ruby-lang.org/es/.

[26] **Jim Weirich.** Rake. [Online] http://rubyforge.org/projects/rake.

[27] The Apache Ant Project. [Online] http://ant.apache.org/.

[28] RDoc - Documentation from Ruby Source Files. [Online] http://rdoc.sourceforge.net/.

[29] **Wikipedia.** Lenguaje de programación interpretado. [Online] http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_interpretado.

[30] —. Comparison of web application frameworks. [Online] http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks.

[31] **Hansson, David Heinemeier.** Ruby on Rails. *Ruby on Rails.* [Online] http://rubyonrails.org/.

[32] **Sun Microsistems.** MySQL. [Online] http://www.mysql.com/.

[33] Mongrel. [Online] http://mongrel.rubyforge.org/.

[34] Subversion. [Online] http://subversion.tigris.org/.

[35] TortoiseSVN. [Online] http://tortoisesvn.tigris.org/.

[36] **Martin Fowler.** Active Record. [Online] http://martinfowler.com/eaaCatalog/activeRecord.html.

[37] Api Ruby on Rails. [Online] http://api.rubyonrails.org/.

[38] **W3C.** World Wide Web Consortium (W3C). [Online] http://www.w3.org/.

[39] **technoweenie.** Restful Authenticated. [Online] http://github.com/technoweenie/restful-authentication.

[40] Nokogiri. [Online] http://github.com/tenderlove/nokogiri.

[41] **W3C.** XPATH. [Online] http://www.w3schools.com/XPath/default.asp.

[42] **Matsumoto, Yukihiro.** Ruby API - Base64. *Ruby-doc - Base64.* [Online] http://ruby-doc.org/core/classes/Base64.html.

[43] **Glazebrook, John.** Open Flash Chart. [Online] http://teethgrinder.co.uk/open-flash-chart-2/.

[44] Javascript Object Notation. [Online] http://www.json.org/.

[45] **Mircrosistems, Sun.** MySQL Migration Toolkit. [Online] http://dev.mysql.com/doc/migration-toolkit/en/index.html.

[46] **MINISTERIO DE SANIDAD Y POLÍTICA SOCIAL.** *Estrategia en EPOC del Sistema Nacional de Salud Aprobada por el Consejo Interterritorial del Sistema Nacional de Salud 3 de junio de 2009.* 2009.

[47] **Shaw, Zed A.** Mongrel. *Mongrel.* [Online] http://mongrel.rubyforge.org/.

[48] **Foundation, The Apache Software.** Apache. *Apache HTTP Server Project.* [Online] http://httpd.apache.org/.

[49] **World Wide Web Consortium (W3C).** XML. *Extensible Markup Language (XML).* [Online] http://www.w3.org/XML/.

[50] **Sanchis, J; Casan, P; Castillo, J; González, N; Palenciano, L; Roca, J.** *Normativa para la espirometría forzada.* s.l. : La Sociedad Española de Neumología (SEPAR), 1985.